



#5

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: G. Kondo

Date: October 21, 2002

Serial No.: 10/064,751

Docket No.: JP920000461US1

Filed: August 13, 2002

Group Art Unit: 2123

For: APPLICATION EDITING APPARATUS
AND DATA PROCESSING METHOD AND
PROGRAM

Assistant Commissioner for Patents
Washington, D.C. 20231

SUBMISSION OF PRIORITY DOCUMENT

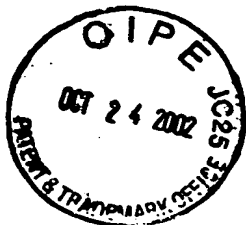
Sir:

Enclosed herewith is a certified copy of Japanese Application No. 2001-246290
filed August 14, 2001, in support of applicant's claim to priority under 35 U.S.C. 119.

Respectfully submitted,

Derek S. Jennings
Reg. Patent Agent/Patent Engineer
Reg. No. 41,473
(914) 945-2144

IBM CORPORATION
Intellectual Property Law Dept.
P. O. Box 218
Yorktown Heights, N. Y. 10598



日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application:

2001年 8月14日

出 願 番 号

Application Number:

特願2001-246290

[ST.10/C]:

[JP2001-246290]

出 願 人

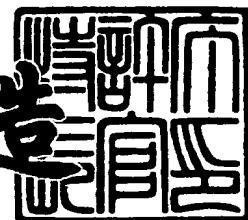
Applicant(s):

インターナショナル・ビジネス・マシーンズ・コーポレーション

2002年 3月15日

特 許 庁 長 官
Commissioner,
Japan Patent Office

及 川 耕 造



出証番号 出証特2002-3017554

【書類名】 特許願

【整理番号】 JP9000461

【提出日】 平成13年 8月14日

【あて先】 特許庁長官 殿

【国際特許分類】 G06F 7/00

【発明者】

 【住所又は居所】 神奈川県大和市下鶴間 1 6 2 3 番地 1 4 日本アイ・ピー・エム株式会社 東京基礎研究所内

 【氏名】 近藤 豪

【特許出願人】

 【識別番号】 390009531

 【氏名又は名称】 インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】

 【識別番号】 100086243

 【弁理士】

 【氏名又は名称】 坂口 博

【代理人】

 【識別番号】 100091568

 【弁理士】

 【氏名又は名称】 市位 嘉宏

【代理人】

 【識別番号】 100106699

 【弁理士】

 【氏名又は名称】 渡部 弘道

【復代理人】

 【識別番号】 100104880

 【弁理士】

 【氏名又は名称】 古部 次郎

【選任した復代理人】

【識別番号】 100100077

【弁理士】

【氏名又は名称】 大場 充

【手数料の表示】

【予納台帳番号】 081504

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9706050

【包括委任状番号】 9704733

【包括委任状番号】 0004480

【ブルーフの要否】 要

【書類名】 明細書

【発明の名称】 アプリケーション編集装置、データ処理方法及びプログラム

【特許請求の範囲】

【請求項 1】 コンピュータを用いて、モデルとビューとを別個に有するアプリケーションの編集を行うアプリケーション編集装置において、

前記アプリケーションにおける第 1 のモデルを編集する編集手段と、

前記編集手段にて編集された第 1 のモデルを第 2 のモデルに変換するモデル変換手段と、

前記第 2 のモデルを当該第 2 のモデルのビューにてディスプレイ装置に表示するビュー表示手段とを備え、

前記ビュー表示手段は、

前記編集手段による前記第 1 のモデルの編集に基づいて前記第 2 のモデルが更新された場合に、当該第 2 のモデルの更新に基づくイベントを生成するイベント生成手段を備え、

前記イベント生成手段にて生成されたイベントに基づいて前記ディスプレイ装置に表示されたビューを変更することを特徴とするアプリケーション編集装置。

【請求項 2】 前記ビュー表示手段は、前記編集手段による前記第 1 のモデルの編集に基づいて前記第 2 のモデルが更新された場合に、当該第 2 のモデルの更新前後の差分を抽出する差分抽出手段をさらに備え、

前記イベント生成手段は、前記差分抽出手段にて抽出された前記差分に関する情報をパラメータとして前記イベントを生成することを特徴とする請求項 1 に記載のアプリケーション編集装置。

【請求項 3】 前記モデル変換手段は、前記第 1 のモデルを要素ごとに前記第 2 のモデルの対応要素に変換することを特徴とする請求項 1 に記載のアプリケーション編集装置。

【請求項 4】 前記モデル変換手段は、前記第 1 のモデルにおける要素の変換結果に対応する要素が前記第 2 のモデルに存在しない場合に、当該変換結果に対応する要素を当該第 2 のモデルに付与することを特徴とする請求項 1 に記載の

アプリケーション編集装置。

【請求項 5】 前記モデル変換手段は、前記第 1 のモデルにおける前記編集手段により編集された要素を、前記第 2 のモデルにおける対応する要素に変換し、変換された当該要素を用いて前記第 2 のモデルを更新することを特徴とする請求項 1 に記載のアプリケーション編集装置。

【請求項 6】 コンピュータを用いて、モデルとビューとを別個に有するアプリケーションの編集を行うアプリケーション編集装置において、

前記アプリケーションにおける第 1 のモデルを編集する編集手段と、

前記編集手段にて編集された第 1 のモデルを第 2 のモデルに変換するモデル変換手段と、

前記第 2 のモデルを当該第 2 のモデルのビューにてディスプレイ装置に表示するビュー表示手段と、

前記第 1 のモデルの更新を当該第 1 のモデルのビューに反映させるためのイベントを、前記第 1 のモデルを前記第 2 のモデルに変換するための変換ルールを用いて、前記第 2 のモデルのビューを変更するイベントに変換するイベント変換手段とを備え、

前記ビュー表示手段は、前記イベント変換手段にて生成されたイベントに基づいて、前記ディスプレイ装置に表示されたビューを変更することを特徴とするアプリケーション編集装置。

【請求項 7】 コンピュータを用いて、所定のアプリケーションにおけるモデルを他のアプリケーションにおけるビューにて表示するデータ処理方法において、

前記所定のアプリケーションにおける第 1 のモデルが更新された場合に、前記アプリケーションを格納したデータ記憶手段から前記他のアプリケーションにおける第 2 のモデルを読み出し、当該更新内容を反映させて当該第 2 のモデルを更新するステップと、

前記第 2 のモデルの更新に基づくイベントを生成し、当該イベントに基づいてディスプレイ装置に表示された前記他のアプリケーションにおけるビューを変更するステップと

を含むことを特徴とするデータ処理方法。

【請求項 8】 前記他のアプリケーションにおけるビューを変更するステップは、

前記第 2 のモデルにおける更新前後の差分を抽出するステップと、

更新前の前記第 2 のモデルに前記差分に相当する変更を加えることにより前記イベントを生成するステップと、

当該イベントに基づいて前記ビューを変更するステップと

を含むことを特徴とする請求項 7 に記載のデータ処理方法。

【請求項 9】 前記第 2 のモデルを更新するステップは、前記第 1 のモデルを要素ごとに前記第 2 のモデルの対応要素に変換するステップを含み、

前記他のアプリケーションにおけるビューを変更するステップは、前記第 2 のモデルにおける変換された要素ごとに当該第 2 のモデルにおける更新前後の差分を抽出するステップを含むことを特徴とする請求項 8 に記載のデータ処理方法。

【請求項 10】 前記他のアプリケーションにおけるビューを変更するステップは、

前記第 1 のモデルの更新を前記所定のアプリケーションにおけるビューに反映させるためのイベントを、前記第 1 のモデルを前記第 2 のモデルに変換するための変換ルールを用いて、前記他のアプリケーションにおけるビューを変更するイベントに変換するステップを含むことを特徴とする請求項 7 に記載のデータ処理方法。

【請求項 11】 コンピュータを制御して、モデルとビューとを別個に有するアプリケーションを実行するプログラムにおいて、

前記アプリケーションを格納したデータ記憶手段から前記アプリケーションにおけるモデルを読み出し、当該モデルのビューをディスプレイ装置に表示する処理と、

前記モデルが更新された場合に、当該モデルの更新前後の差分を抽出する処理と、

抽出された前記差分に基づいて前記ビューを変更するためのイベントを生成する処理と、

生成された前記イベントに基づいて前記ディスプレイ装置に表示されている前記ビューを変更する処理と

を前記コンピュータに実行させることを特徴とするプログラム。

【請求項 1 2】 コンピュータを制御して、モデルとビューとを別個に有するアプリケーションを編集するプログラムにおいて、

前記アプリケーションにおける第 1 のモデルに対する編集を行う編集手段と、

前記編集手段にて編集された第 1 のモデルを第 2 のモデルに変換するモデル変換手段と、

前記モデル変換手段により前記第 1 のモデルが前記第 2 のモデルに変換された場合に、当該第 2 のモデルと直前に変換された前記第 2 のモデルとの差分を抽出する差分抽出手段と、

前記差分抽出手段にて抽出された差分に基づくイベントを生成するイベント生成手段と、

前記第 2 のモデルを当該第 2 のモデルのビューにてディスプレイ装置に表示すると共に、前記イベント生成手段にて生成されたイベントに基づいて前記ディスプレイ装置に表示されたビューを変更するビュー表示手段として、

前記コンピュータを動作させることを特徴とするプログラム。

【請求項 1 3】 前記モデル変換手段は、前記第 1 のモデルを要素ごとに前記第 2 のモデルの対応要素に変換することを特徴とする請求項 1 2 に記載のプログラム。

【請求項 1 4】 前記差分抽出手段は、前記第 2 のモデルにおける変換された要素ごとに当該第 2 のモデルにおける更新前後の差分を抽出することを特徴とする請求項 1 2 に記載のプログラム。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

本発明は、モデルオブジェクトとビューオブジェクトとを別個に有するアプリケーションにおけるモデル変換及びビューの表示方法に関する。

【0 0 0 2】

【従来の技術】

アプリケーションプログラム（以下、アプリケーション）を対話的に作成する際の基本的な手法として、アプリケーションのロジックであるモデルオブジェクト（以下、モデル）と、ディスプレイ装置に対する当該アプリケーションの表示態様であるビューオブジェクト（以下、ビュー）とに分割する方法がある。アプリケーションをモデルとビューとに分割することにより、1つのモデルに対して、種々のビューを対応づけることができる。すなわち、1つのプログラムを処理や操作に応じた種々の表示態様で利用することが可能となる。

【0003】

図36は、モデルとビューとの関係を例示する図である。

図36に示す例では、1つのモデル（ツリー構造）に対してWysiwyg表示、ツリー表示、ソース表示の3種類のビューが与えられている。すなわち、ユーザは、図示のモデルで特定されるアプリケーションを、閲覧や編集といった用途に応じてWYSIWYG（What You See Is What You Get）表示、ツリー表示、ソース表示の中から任意のビューを選んで表示することができる。

【0004】

このようなアプリケーションでは、所定のビューを通してモデルの変更が加えられた場合、その変更が他のビューに対しても反映される。

図37は、図36のアプリケーションにおいて、所定のビューにおける変更がモデル及び他のビューに反映する様子を示す図である。

図37において、WYSIWYG表示のビューに書き込みが行われると、次の手順で他のビューに反映される（手順の数字は図中の数字に対応）。

1. ビューに書き込みが発生。
2. 変更が発生したビューオブジェクトに対応するモデルオブジェクトを変更（図示の例ではノードの追加）。
3. 変更内容に対応するイベントを生成して各ビューに同報。
4. 各ビューが受け取ったイベントを解釈して更新。

【0005】

ところで、上記のようにモデルとビューとを分割したアプリケーションでは、

1つのモデルに対して複数のビューを自由に追加削除できるという利点がある。ただし、所定のモデルに対して所定のビューを使用するためには、モデルの内容に対応した表示を行ったり、モデルに変更があった場合に生成される上記のイベントを適切に処理して反映させたりするために、当該ビューは、当該モデルに対応した一定のインターフェイスに合致させたものである必要がある。

所定のモデルに対し所定のビューを追加しようとする場合で、かつ当該モデルに対応したインターフェイスを持ったビューが存在しない場合を考える。これを解決するために、幾つかの方法が考え得る。最も直接的な方法は、要求されるインターフェイスを持った必要なビューを作成する方法である。ただし、所望のモデルに対して所望のビューをいちいち作成するとすれば、作業コストがかかりすぎるという問題がある。

【0006】

そこで、当該モデルにおいて要求されるインターフェイスとは異なる、他のアプリケーションで使用されるインターフェイスを持ったビューが存在する場合、このビューを当該モデル用に使用することが考えられる。Adapterパターンとして知られる手法などを使ってインターフェイスを適応させることにより、当該ビューを所望のモデルに対して使用することができる。Adapterパターンについては、例えば次の文献に詳細に記述されている。

文献：Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns, 1995.

【0007】

図38は、Adapterパターンを模式的に例示する図である。

図38(A)に示すように、アプリケーションAとアプリケーションBが、それぞれ独自のインターフェイスに基づいたモデルとビューの対を持っているものとする。この場合、アプリケーションBのモデルBを表示するためにアプリケーションAのビューAを使用することはできない。しかしながら、図38(B)に示すように、アプリケーションA、Bのインターフェイスを適合させるAdapter

を介在させることにより、モデル B を表示するためにビュー A を用いることが可能となる。

【 0 0 0 8 】

また、他のアプリケーションで使用されるビューを使用するための他の方法として、モデル変換器を用いて所望のモデルの内容を当該他のアプリケーションにおけるモデルに反映させ、当該他のアプリケーションにおけるビューで表示する方法が考えられる。

図 3 9 は、モデル変換により他のアプリケーションのビューを用いて表示を行う方法を説明する図である。

図 3 9 に示すように、アプリケーション A にはモデル A を表示するために W Y S I W Y G 表示のビュー A 1 及びツリー表示のビュー A 2 があり、アプリケーション B にはモデル B を表示するためにソース表示のビュー B 1 及びツリー表示のビュー B 2 があり、アプリケーション C にはモデル C を表示するためにソース表示のビュー C がある。ここで、アプリケーション A において、ソース表示を行いたい場合、アプリケーション A にはソース表示を行うビューがない。そこで、モデル A を、モデル変換器を用いてモデル B またはモデル C に変換し、ビュー B 1 またはビュー C を用いてソース表示を行う。同様に、アプリケーション B やアプリケーション C で W Y S I W Y G 表示を行いたい場合や、アプリケーション C でツリー表示を行いたい場合、それぞれのビューを持ったアプリケーションのノードに変換することによって、当該変換先のアプリケーションが持つビューを利用することができる。

【 0 0 0 9 】

上記のモデル変換器の代表的な例として X S L T (XSL Transformations) プロセッサがある。X S L T において、モデル変換を行うためのルールは、X S L (eXtensible Style Language) で記述される。

X S L T プロセッサの代表的な実装に、A p a c h e プロジェクトから提供されている X a l a n がある。X a l a n は、ストリーム以外に XML における上記モデルである D O M (Document Object Model) を入力として変換することができる。この D O M の変換は、変換元であるモデルにおける内容の変更を即時に

反映させるもの（以下、このような変換を動的な変換と称す）ではなく、1つのモデルを全体で他のモデルに変換するもの（以下、このような変換を静的な変換と称す）である。すなわち、変換元のモデルにおける変更を変換先のモデルに反映させるには、変更が行われた変換元のモデル全体を一括して変換することによって行う。X a l a n 以外にも種々の X S L T プロセッサが存在するが、そのほとんどは、X a l a n と同様の静的な変換を行うものである。

【 0 0 1 0 】

米国マイクロソフト社より提供されている XML モジュールである M S X M L に付属された X S L T プロセッサは、入力である変換元のモデルは静的であって更新がないが、変換ルールを変更することができる。これによって、変換先のモデルとビューを動的に変えることが可能となる。しかしながら、これは実際には静的な変換をルールごとに逐次行っているに過ぎず、変換元の動的な変更には対応していない。

【 0 0 1 1 】

また、TFI Technologyから提供されている N a p a は、入力をストリームとして受け取り、それをプログレッシブに変換して、変換結果を出力ストリームに与える X S L T プロセッサである。入力ストリームから読まれた変換元モデルにデータが追記されていくという意味では、変換元が動的に変化しているとも言えるが、変換そのものはストリームからストリームへの静的な変換であり、変換元の動的な変更に対応しているわけではない。

【 0 0 1 2 】

さらに、モデル変換器は単体で使われるだけでなく、エディタに組み込まれたプレビュー用モデルの生成にも応用されている。そのようなモデル変換器の機能を持ったエディタとしては、Wattle Softwareから提供されている XML w r i t e r や、eXcelonから提供されている E x c e l o n S t y l u s などがある。これらのエディタでは、変換元のモデルの編集用に、ソースコードのビュー（ソース表示）によって表示する。ユーザが XML データの編集に、当該編集に基づく更新されたプレビュー画面を参照したい場合は、明示的な操作によってプレビュー画面の更新機能呼び出す。これに応じて、エディタに搭載された

モデル変換器が、変換元モデルを一括して変換して変換先モデルを新たに作成し、当該変換先モデルのビューであるプレビュー画面を更新する。

【 0 0 1 3 】

【発明が解決しようとする課題】

上述したように、所定のモデルに対し所定のビューを追加しようとする場合であって、かつ当該モデルに対応したインターフェイスを持ったビューが存在しない場合の解決手段は幾つか考えられるが、要求されるインターフェイスを持った必要なビューを作成する方法は、多大な作成コストを要するという問題があった。

【 0 0 1 4 】

また、Adapterパターンなどを用い、インターフェイスの異なるビューをモデルに適応させて使用する方法は、ビュー自体を作成するほどのコストは要しないが、モデル・ビュー間のインターフェイスは、アプリケーションによっては複雑であり、narrowインターフェイス、つまり適応しなければならないオペレーションの最小の集合が大きい場合、Adapterパターンの作成には、やはり多大な作業コストを要することとなっていた。

【 0 0 1 5 】

さらに、モデル変換器を用いて他のモデルに対応したビューを使用する方法では、モデル変換器を作成する作業コストは、インターフェイスの異なるビューをモデルに適応させる方法と比して低い。

しかし、モデル変換器によるモデルの変換は静的な変換であるため、変換元のモデルに変更が加えられた場合に、即時的に変換先のモデルを変更し、当該変更をビューに反映させることができなかった。すなわち、変換元のアプリケーションでは、変換先のアプリケーションにおいて、モデルがどのようなイベントを発生させることによってビューを変更しているのか知ることができないため、変換元のモデルにおける変更部分のみを変換先のビューに反映させる更新を行うことができない。したがって、変換元のモデルを一括して変換先のモデルに変換し、さらに当該新たな変換先のモデルに基づいてビューを作成し直すという過程を経ていた。

【 0 0 1 6 】

XML writerやExcelon Stylusなどのエディタは、上述したように、アプリケーションのモデルの他にプレビュー用のモデルを作成し、当該プレビュー用のモデルに対応したビューを用いてプレビュー画面を表示している。しかし、アプリケーションで行われた編集作業によりモデルが変更された場合、変換先であるプレビュー用のモデル及びビュー（以下、このモデルとビューとを併せてモデル・ビュー対と称す）に対して即時的に変更が反映されるのではなく、ユーザが明示的に変換先の更新を要求することによって、一括して変換元のモデルを変換先のモデル・ビュー対に変換している。

すなわち、変換先のモデル・ビュー対が変更された場合であっても、変換先におけるモデルからビューにイベントが伝えられて更新された結果、変更後の新しいビューになっているわけではない。したがって、この場合、変更前のモデル・ビュー対は破棄され、更新後の変換元モデルに基づいて生成された、新しい変換先モデル・ビュー対を作り直すといった、効率の悪い手法となっている。

特に変換元モデル、変換先モデルの一方あるいは両方においてデータサイズが大きい場合、プレビューを要求するたびに処理に多大な時間を要することとなる。また、ハードウェアにおいてもメモリ上の大量の記憶空間を消費する。

【 0 0 1 7 】

さらにまた、変換先のモデル・ビュー対において、変換元のモデルにおける変更の内容のみを反映させるのではなく、モデル・ビュー対を新たに作り直すため、変換先のモデル・ビュー対において、編集過程で生じたカーソルの位置や選択領域の位置などのデータを保持している場合、モデル・ビュー対を作り直すたびにこれらの情報が失われることとなる。

【 0 0 1 8 】

これらの課題に対応する手段として、変換元のモデルの変更を変換先のモデルに反映させ、当該変換先のモデルにおいて当該更新を実現するイベントを生成し、このイベントによって変換先のビューを更新する手段が考えられる。

しかしながら、変換元のモデル・ビュー対と変換先のモデル・ビュー対とでは、通常、イベントの定義自体が異なるため、変換元モデルの更新から変換先ビュ

ーを更新するためのイベントを発生させることができない。

【0019】

そこで、本発明は、モデル変換を用いて所定のモデルから他のアプリケーションのビューを使用する場合に、変換元のモデルに対する変更を動的に変換先のモデル及びビューに反映させ、ビューを更新できるようにすることを目的とする。

【0020】

また、本発明は、モデル変換を用いて所定のモデルから他のアプリケーションのビューを使用する場合に、変換元のモデルの変更に対応する部分的な変更を変換先のビューに対して行うことによりビューを更新するシステムを提供することを他の目的とする。

【0021】

【課題を解決するための手段】

上記の目的を達成する本発明は、コンピュータを用いて、モデルとビューとを別個に有するアプリケーションの編集を行うアプリケーション編集装置において、このアプリケーションにおける第1のモデルを編集する編集手段と、この第1のモデルを第2のモデルに変換するモデル変換手段と、この第2のモデルをこの第2のモデルのビューにてディスプレイ装置に表示するビュー表示手段とを備える。そして、このビュー表示手段は、編集手段による第1のモデルの編集に基づいて第2のモデルが更新された場合に、この第2のモデルの更新に基づくイベントを生成するイベント生成手段を備え、このイベント生成手段にて生成されたイベントに基づいてディスプレイ装置に表示されたビューを変更することを特徴とする。

【0022】

ここで、このビュー表示手段は、編集手段による第1のモデルの編集に基づいて第2のモデルが更新された場合に、この第2のモデルの更新前後の差分を抽出する差分抽出手段をさらに備える構成とすることができる。この場合、イベント生成手段は、差分抽出手段にて抽出された差分に関する情報をパラメータとしてイベントを生成する。

【0023】

また、このモデル変換手段は、第1のモデルを要素ごとに第2のモデルの対応要素に変換する構成とすることができる。

さらに詳しくは、このモデル変換手段は、第1のモデルにおける要素の変換結果に対応する要素が第2のモデルに存在しない場合に、この変換結果に対応する要素を第2のモデルに付与する構成とすることができる。

このように構成すれば、第2のモデルを更新した場合、更新前後で変化のない要素については更新前の要素をそのまま用いることができ、第2のモデルとそのビューとの間で保持されている情報を保存することができる。

【0024】

さらにまた、このモデル変換手段は、第1のモデルにおける編集手段により編集された要素を、第2のモデルにおける対応する要素に変換し、変換された要素を用いて第2のモデルを更新する構成とすることができる。

このような構成とすれば、第1のモデルにおいて変更された要素のみを第2のモデルにおける要素に変換し、第2のモデルに反映させることができるので、モデル変換における作業効率を向上させることができる。

【0025】

また、本発明は、上記のようなアプリケーション編集装置において、アプリケーションにおける第1のモデルを編集する編集手段と、この第1のモデルを第2のモデルに変換するモデル変換手段と、この第2のモデルをこの第2のモデルのビューにてディスプレイ装置に表示するビュー表示手段と、第1のモデルの更新をこの第1のモデルのビューに反映させるためのイベントを、モデル変換に用いられる変換ルールを用いて、第2のモデルのビューを変更するイベントに変換するイベント変換手段とを備え、ビュー表示手段は、このイベント変換手段にて生成されたイベントに基づいて、ディスプレイ装置に表示されたビューを変更する構成とすることができる。

このような構成とすれば、第2のモデルにおいて更新前後の差分を抽出することなく第2のモデルのビューを変更するイベントを生成することができる。

【0026】

さらにまた、本発明は、コンピュータを用いて、所定のアプリケーションにお

けるモデルを他のアプリケーションにおけるビューにて表示するデータ処理方法において、所定のアプリケーションにおける第1のモデルが更新された場合に、データ記憶手段から他のアプリケーションにおける第2のモデルを読み出し、第1のモデルにおける更新内容を反映させて第2のモデルを更新するステップと、この第2のモデルの更新に基づくイベントを生成し、このイベントに基づいてディスプレイ装置に表示された他のアプリケーションにおけるビューを変更するステップとを含むことを特徴とする。

【0027】

詳しくは、他のアプリケーションにおけるビューを変更するステップは、第2のモデルにおける更新前後の差分を抽出するステップと、更新前の第2のモデルに抽出された差分に相当する変更を加えることにより前述のイベントを生成するステップと、このイベントに基づいてビューを変更するステップとを含む。

【0028】

また、第2のモデルを更新するステップは、第1のモデルを要素ごとに第2のモデルの対応要素に変換するステップを含み、かつ他のアプリケーションにおけるビューを変更するステップは、第2のモデルにおける変換された要素ごとにこの第2のモデルにおける更新前後の差分を抽出するステップを含む構成とすることができる。

【0029】

さらに、他のアプリケーションにおけるビューを変更するステップは、第1のモデルの更新をそのビューに反映させるためのイベントを、第1のモデルを第2のモデルに変換するための変換ルールを用いて、他のアプリケーション、すなわち第2のモデルのビューを変更するイベントに変換するステップを含む構成とすることができる。

【0030】

また、本発明は、次のように構成されたプログラムとして実現することができる。すなわち、コンピュータを制御して、モデルとビューとを別個に有するアプリケーションを実行するプログラムであって、このプログラムは、このアプリケーションを格納したデータ記憶手段からこのアプリケーションにおけるモデルを

読み出し、このモデルのビューをディスプレイ装置に表示する処理と、このモデルが更新された場合に、このモデルの更新前後の差分を抽出する処理と、抽出された差分に基づいてビューを変更するためのイベントを生成する処理と、生成されたイベントに基づいてディスプレイ装置に表示されているビューを変更する処理とをこのコンピュータに実行させる。

【 0 0 3 1 】

あるいは、コンピュータを制御して、モデルとビューとを別個に有するアプリケーションを編集するプログラムであって、このプログラムは、このアプリケーションにおける第 1 のモデルに対する編集を行う編集手段と、編集手段にて編集された第 1 のモデルを第 2 のモデルに変換するモデル変換手段と、このモデル変換手段により第 1 のモデルが第 2 のモデルに変換された場合に、この第 2 のモデルと直前に変換された第 2 のモデルとの差分を抽出する差分抽出手段と、この差分抽出手段にて抽出された差分に基づくイベントを生成するイベント生成手段と、第 2 のモデルをそのビューにてディスプレイ装置に表示すると共に、このイベント生成手段にて生成されたイベントに基づいてディスプレイ装置に表示されたビューを変更するビュー表示手段として、このコンピュータを動作させる。

【 0 0 3 2 】

【発明の実施の形態】

以下、添付図面に示す本実施の形態について本発明を詳細に説明する。

本発明では、モデルとビューとを別個に有するアプリケーションにおいて、モデルの表示を行うために、モデル変換によって他のアプリケーションのビューを使用する手法を対象とする。そして、変換元のモデルに変更があった場合に、モデル変換によって、当該変更を変換先のモデルに反映させる。この後、当該変換先のモデルにおける変更内容に基づいてビューを更新させるためのイベントを生成する。このイベントによって、ビューは、変換元のモデルにおいて変更のない部分についてはそれまでの表示内容を保持しつつ、変換元のモデルの変更に対応する部分の変更を行うといった更新を実現することができる。

【 0 0 3 3 】

〔実施の形態 1〕

本実施の形態は、変換元のモデルに変更があった場合に、まず変換元のモデルを変換先のモデルに一括して変換し、次に変換先のモデルにおいて変更前後の差分を抽出して変更内容を特定する。そして、当該変更内容に応じたイベントを生成してビューに送り、当該変更内容をビューに反映させることにより、アプリケーションの編集におけるビューの動的な変更を実現する。

【 0 0 3 4 】

図 1 は、本実施の形態によるアプリケーション編集装置の実現に好適なコンピュータ装置の構成例を模式的に示した図である。

図 1 に示すコンピュータ装置は、CPU（中央処理装置）101と、システムバスを介してCPU101に接続されたM/B（マザーボード）チップセット102及びメインメモリ103と、PCIバスなどの高速なバスを介してM/Bチップセット102に接続されたビデオカード104、ハードディスク105及びネットワークインターフェイス106と、さらにブリッジ回路110及びISAバスなどの低速なバスを介してM/Bチップセット102に接続されたフロッピーディスクドライブ107、キーボード108及びI/Oポート109とを備える。

【 0 0 3 5 】

なお、図 1 は、本実施の形態によるアプリケーション編集装置としてのコンピュータ装置の構成を例示するに過ぎず、本実施の形態を適用可能であれば、他の種々のシステム構成を取ることが可能である。例えば、ビデオカード104の代わりにビデオメモリのみを設け、CPU101自身が描画命令を実行するようなシステムとしても良い。また、図示はしないが、一般的なコンピュータ装置には、入力手段としてのマウス、音声入出力機構、CD-ROMドライブなどが接続される。

【 0 0 3 6 】

図 2 は、プログラム制御されたCPU101において、本実施の形態によるアプリケーションの編集を行うための機能ブロックを示す図である。

図 2 を参照すると、本実施の形態は、アプリケーションAを実行するアプリケーションA実行部10と、アプリケーションAにおけるモデルAをアプリケーシ

ョンBにおけるモデルBに変換するモデル変換器20と、アプリケーションBを実行するアプリケーションB実行部30とを備える。

なお、図2に示すアプリケーションA実行部10、モデル変換器20及びアプリケーションB実行部30は、図1のメインメモリ103にロードされたコンピュータプログラムにより制御されたCPU101にて実現される仮想的なソフトウェアブロックである。当該コンピュータプログラムは、CD-ROMやフロッピーディスクなどの記憶媒体に格納して配布したり、ネットワークを介して伝送したりすることにより提供することができる。このようにして提供されたプログラムが、フロッピーディスクドライブ107や図示しないCD-ROMドライブ、ネットワークインターフェイス106などを介してメインメモリ103に読み込まれ、CPU101を制御することにより、図1に示したコンピュータ装置が図2に示す各機能を実現する。

【0037】

ここで、アプリケーションAは変換元となるアプリケーションを示し、アプリケーションBは変換先となるアプリケーションを示すに過ぎない。すなわち、個々のアプリケーションが、変換元として扱われる場合はアプリケーションAとなり、変換先として扱われる場合はアプリケーションBとなる。

図3は、アプリケーションAとアプリケーションBとの関係を説明する図である。

図3に示すように、アプリケーションAは、ツリー構造のモデルAとWYSIWYG表示のビューAとを持つものとする。また、アプリケーションBは、ツリー構造のモデルBとツリー表示のビューBとを持つものとする。本実施の形態では、アプリケーションAにおけるモデルAを、アプリケーションBにおけるビューBを用いて表示するためにモデル変換を行う。なお、アプリケーションA、Bにおけるビューは1種類である必要はないが、ここでは簡単のため、それぞれ1種類のビューA、Bを持っているものとする。

【0038】

図2の構成において、アプリケーションA実行部10は、図1のメインメモリ103に読み込まれたアプリケーションAを実行し、ビデオカード104を介し

てディスプレイ装置にモデルAに基づくビューAを表示する。また、アプリケーションA実行部10は、モデルAの編集手段として、キーボード108等を介してユーザがビューAに対して行った編集操作を受け付けて、モデルAに反映させる。さらに、この編集操作によってモデルAに対して変更があった場合は、変更があったことをモデル変換器20に知らせる。

【0039】

モデル変換器20は、アプリケーションAのモデルAをアプリケーションBのモデルBに変換する。モデル変換の方法としては、従来のモデル変換器による方法と同様の方法を用いることができる。アプリケーションA実行部10において、モデルAの内容が変更された場合は、変更の通知を受けて、当該変更後のモデルAを入力し、当該変更内容を反映させたモデルBに変換する。

【0040】

アプリケーションB実行部30は、図1のメインメモリ103に読み込まれたアプリケーションBを実行し、ビデオカード104を介してディスプレイ装置にモデルBに基づくビューBを表示する。

また、アプリケーションB実行部30は、モデルBが更新された場合に、その更新内容に基づいてビューBを更新する。ここで、ビューBの更新は、モデルBの更新前後の差分に基づいて行う。これを実現するため、アプリケーションB実行部30は、差分抽出部31とイベント生成部32とを備える。

【0041】

上記のように、アプリケーションA実行部10においてモデルAが変更されると、モデル変換器20により、当該モデルAの変更内容を反映させてモデルBが更新される。これに応じて、アプリケーションB実行部30の差分抽出部31は、当該更新後のモデルBと更新前のモデルBとを比較して、差分を抽出する。

イベント生成部32は、差分抽出部31によりモデルBの更新前後の差分が抽出されると、当該差分をパラメータとしてイベントを作成する。すなわち、更新前のモデルBに対して当該差分を満たす変更を行い、この変更をビューBに反映させるためのイベントを作成する。

【0042】

アプリケーションB実行部30は、イベント生成部32にて作成されたイベントをビューBに送り、ビューBを更新する。これによって、ビューBは、変更のない部分については更新前の表示内容をそのまま表示し、変更部分についてはノードAの変更内容を反映させた表示内容となるように更新される。

また、アプリケーションB実行部30は、当該変更がなされたモデルBを保持することとなるが、この場合、イベント生成部32においてイベントを作成するために行われた変更結果を保持しても良いし、モデル変換器20による変換において生成されたモデルBを保持しても良い。イベント生成部32にて変更されたモデルBを保持する場合、モデル変換器20にて生成されたモデルBを破棄する無駄が生じるが、更新前のモデルB及びビューBの対において、編集過程で生じたカーソルの位置や選択領域の位置などのデータを保持している場合、この情報を保存することができる。

【0043】

図4は、上述したアプリケーションA実行部10、モデル変換器20、アプリケーションB実行部30によるモデル及びビューの更新の動作を説明するフローチャートである。また、図5は、かかる更新動作を図3のモデル・ビュー対に対して適用した様子を示す図である。図5中に示された符号は、図4のステップに対応する。

図4に示すように、変換元のビューAに書き込みが発生すると（ステップ401）、アプリケーションA実行部10により、変更されたビューAに対応してモデルAが変更される（ステップ402）。そして、アプリケーションA実行部10からモデル変換器20へ、モデルAの更新が通知される。

次に、アプリケーションA実行部10からの通知を受け取ったモデル変換器20が、モデルAにおける変更を反映させたモデルBを生成し、アプリケーションB実行部30に渡す（ステップ403）。

【0044】

アプリケーションB実行部30は、モデル変換器20からモデルAの変更を反映させたモデルBを受け取り、差分抽出部31により、モデルBの更新前後の差分を抽出する（ステップ404）。そして、イベント生成部32により、この差

分をパラメータとするイベントを生成する（ステップ405）。このイベントによって、ビューBが、モデルAの変更内容を反映させた表示に更新される（ステップ406）。

【0045】

上述した更新前後のモデルBの差分を用いてビューBを動的に変更する手法を用いれば、従来のように更新されたモデルBに基づいてビューBを作り直す静的な更新に比べて、更新における処理速度が向上する。これにより、アプリケーションA実行部10におけるビューAに対する編集操作に対し、即時的にビューBを更新させることが可能となり、ビューBを確認しながらの編集作業が可能となる。

【0046】

ここで、ビューBの更新における処理速度の向上を、処理コストの観点から検討する。

本実施の形態において、モデルAの変更が発生してからビューBが更新されるまでのコストVt1は、以下のように定義される。

$$Vt1 = Vconv1 + Vdiff1 + Vupdate + Const \quad (1)$$

式（1）において、

Vconv：モデルの変換コスト。変換元モデル（モデルA）のエレメント数をmとすると計算量とメモリ空間使用量はO（m）。

Vdiff：差分作成コスト。変換先モデル（モデルB）のエレメント数をnとすると計算量はO（n）。

Vupdate：変換先のビュー（ビューB）によるイベントの解釈及び再描画に要するコスト。変換先モデル（モデルB）のエレメント数をnとすると計算量はO（n）。

Const：その他（ステップ401、402、405など）の処理コスト。モデルのエレメント数に依存しないのでO（1）。

なお、O（m）、O（n）は、それぞれm、nの定数倍以内の時間で、O（1

) は 1 が定数なので定数時間内で計算できることを意味する。

【 0 0 4 7 】

一方、従来技術で用いられるような、更新後のモデル B を反映したビュー B を改めて作り直す手法の場合、ビュー B の再描画がなされるまでのコスト V_{t0} は、以下のように定義される。

$$V_{t0} = V_{conv} + V_{view} + V_{draw} + Const \quad (2)$$

式 (2) において、

V_{view} : 変換先ビュー B の作成コスト。メモリ空間使用量、計算量ともに $O(n)$ 。

V_{draw} : 変換先ビュー B の初期描画コスト。計算量は $O(n)$ 。

【 0 0 4 8 】

上の定義式 (1) (2) から、 V_{t0} と V_{t1} の比較は、 $(V_{diff1} + V_{update})$ と $(V_{view} + V_{draw})$ との比較になる。ここで、

$$V_{update} \ll V_{init}$$

という関係が成り立つ。なぜなら、既に初期描画がなされておりイベントを解釈して当該イベントに関わりのある領域を再描画する方が、ビュー B の全ての領域を描画し直すよりも圧倒的に速いからである。よって、 V_{t0} と V_{t1} との比較は、 V_{diff1} と V_{view} との比較に帰着できるが、計算量のオーダーは同じであり正確には優劣が判断できない。しかし、少なくともメモリ空間使用量の点では、 V_{diff1} の方が優れていることがわかる。

【 0 0 4 9 】

〔実施の形態 2〕

上述した実施の形態 1 では、モデル変換を行う際に、変換先モデル (モデル B) 全体を作成している。しかしながら、ビュー B の更新に必要な情報はモデル B における更新前後の差分であるため、その他の部分の変換は冗長な処理となって

いる。モデルがツリーなどの構造を持ったオブジェクトの集まりである場合、変更のない個所は変更前のオブジェクトをそのまま再利用したほうが、モデル変換における効率が良い。

そこで、本実施の形態では、モデル変換においても、変換元モデル（モデルA）の変更部分に限定して処理を行うこととする。

【0050】

本実施の形態は、実施の形態1と同様に、図1に示すようなコンピュータ装置にて実現される。

図6は、本実施の形態によるアプリケーションの編集を行うための機能ブロックを示す図である。

図6を参照すると、本実施の形態は、アプリケーションAを実行するアプリケーションA実行部10と、アプリケーションAにおけるモデルAをアプリケーションBにおけるモデルBに変換するモデル変換器40と、アプリケーションBを実行するアプリケーションB実行部30とを備える。

これらの構成要素のうち、アプリケーションA実行部10及びアプリケーションB実行部30は、図2に示したアプリケーションA実行部10及びアプリケーションB実行部30と同様であるため、同一の符号を付して説明を省略する。また、アプリケーションAとアプリケーションBとの関係は、実施の形態1において図3を参照して説明したものと同様である。

なお、図6に示すモデル変換器40は、図2に示した各構成要素と同様に、図1のメインメモリ103にロードされたコンピュータプログラムにより制御されたCPU101にて実現される仮想的なソフトウェアブロックである。

【0051】

図6の構成において、モデル変換器40は、モデルAと変更前のモデルBとを入力し、変更後のモデルBを出力する。ただし、最初に変換を行う場合は変更前のモデルBは存在していないので、モデルAのみを入力して通常の変換処理を行う。

図7は、モデル変換器40による変換処理の様子を示す図である。

図7に示すように、モデル変換器40は、入力したモデルAと変更前のモデル

Bとを要素ごとに比較し、モデルAの変更に対応する変更をモデルBに加える。

【 0 0 5 2 】

図7に示した変換処理は、モデルAの要素をモデルBの要素に置き換える操作に分解できる。モデル変換器40は、図6に示すように、この要素の変換を行うための副変換器41を内蔵する。

図8は、副変換器41による変換処理の様子を示す図である。

図8に示すように、副変換器41は、モデルAにおける変換対象の要素と、変更前のモデルBにおける対応部分の要素とを入力し、必要な変更をモデルBの対応部分に加えて出力する。

【 0 0 5 3 】

図9は、モデル変換器40が副変換器41による変換処理を統合してモデル変換を行う処理を説明するフローチャートである。

図9に示すように、モデル変換器40は、モデルAとモデルBとを入力した後、モデルAの要素を1つずつ選択して順次副変換器41の入力1とする（ステップ901）。

副変換器41は、モデルAの要素を入力すると、モデルBのマッピングをたどり、対応する要素を入力2として、さらに入力する（ステップ902）。対応する要素がモデルBに存在しない場合は、「N u l l」を入力2する。

【 0 0 5 4 】

次に、副変換器41は、所定の変換ルールに基づいて入力1であるモデルAの要素をモデルBの対応する要素に変換し、バッファに一時的に格納する（ステップ903）。そして、副変換器41は、入力2が「N u l l」であった場合は、バッファに格納された入力1の変換後の要素を出力とし（ステップ904、905）、「N u l l」でなかった場合は、入力2をバッファにコピーして出力とする（ステップ904、906）。すなわち、モデルAの対応要素がモデルBに存在する場合は当該対応要素が出力とされ、対応要素がモデルBに存在しない場合にステップ903による当該要素の変換結果が出力とされる。

【 0 0 5 5 】

この後、副変換器41は、入力1の要素と出力の要素との間にマッピングを作

成し、当該要素に対する変換処理（要素変換）を終了する（ステップ 9 0 7）。

当該要素変換が終了すると、モデル変換器 4 0 は、未処理の要素があるか調べ、あれば上記の処理を繰り返す（ステップ 9 0 8）。そして、モデル A の全ての要素に対して上記の処理が行われたならば、処理を終了する。

【 0 0 5 6 】

上記のようなモデル変換器 4 0 を用いれば、図 1 の CPU 1 0 1 におけるキャッシュメモリやメインメモリ 1 0 3 における変換作業用の記憶空間の使用量を削減することができる。

また、ステップ 9 0 2 において入力 1 であるモデル A の要素の変換結果をバッファに格納する際に、当該変換結果とモデル B の対応要素とを比較し、これらが異なる場合はその違いをモデル B の更新前後の差分として抽出することにより、アプリケーション B 実行部 3 0 が行う差分抽出のプロセスを簡略化することができる。すなわち、アプリケーション B 実行部 3 0 において変更前後のモデル B を比較して差分を抽出するのではなく、モデル変換器 4 0 における要素変換の際の要素ごとの比較によって抽出された差分を用いて、ビュー B を更新するためのイベントを発生させることができる。

【 0 0 5 7 】

ところで、モデル A の要素とモデル B の要素とは必ずしも 1 対 1 で対応するとは限らないが、要素の個数の増減（モデル A の 1 つの要素にモデル B の複数の要素の組合せが対応する場合、及びモデル A の複数の要素の組合せにモデル B の 1 つの要素が対応する場合）に対しても、上述した副変換器 4 1 を用いた再帰的な処理によって対応することができる。

具体的なモデル変換の例を挙げて説明する。

図 1 0 は、この種のモデル変換を説明する図である。

この例では、モデル A、B が共にツリー構造のオブジェクトとして表現されている場合を取り上げる。このモデル A、B は、ツリーの各ノードに、当該ノードの子ノードを示す「child」という配列型の属性が備わっている。

この場合、モデル A の要素数に対して対応するモデル B の要素数が増減するような変換の種類は、大きく 3 種類に分けられる。すなわち、減少型、増加型、倍

化型の 3 種類である。

【 0 0 5 8 】

まず、減少型について説明する。

減少型の変換の様子を図 1 0 (A) に示す。図 1 1 は、図 1 0 (A) の変換を、要素ごとの処理に分解して示した図である。

図 1 0 (A) 及び図 1 1 を参照すると、まず、モデル A のノード s 1 が、モデル B のノード d 1 に変換される。次に、ノード s 2 に対応するノードがモデル B には存在しないので N u l l となる。そして、モデル A におけるノード s 2 の子孫が、モデル B におけるノード d 1 の子孫に対応づけられる。これにより、次に、ノード s 2 の子孫であるノード s 3、s 4、s 5 が、それぞれモデル B のノード d 2、d 3、d 4 に変換される。

以上の変換を実現する変換ルールを次に示す。

1. s1 -> d1
2. s1.child[0] -> Null
3. s2 -> Null
4. s2.child[0] -> d1.child[0]
5. s2.child[1] -> d1.child[1]
6. s2.child[2] -> d1.child[2]
7. s3 -> d2
8. s4 -> d3
9. s5 -> d4

【 0 0 5 9 】

次に、増加型について説明する。

増加型の変換の様子を図 1 0 (B) に示す。図 1 2 は、図 1 0 (B) の変換を、要素ごとの処理に分解して示した図である。

図 1 0 (B) 及び図 1 2 を参照すると、まず、モデル A のノード s 6 が、モデル B のノード d 5、d 6 に変換される。ここで、ノード d 5 の child 属性として

ノード d 6 が記述される（これによりノード d 6 がノード d 5 の子ノードであることが示される）。そして、モデル A におけるノード s 6 の子孫が、モデル B におけるノード d 6 の子孫に対応づけられる。これにより、次に、ノード s 6 の子孫であるノード s 7、s 8、s 9 が、それぞれモデル B のノード d 7、d 8、d 9 に変換される。

以上の変換を実現する変換ルールを次に示す。

1. s6 -> d5, d6, d5.child[0]=d6
2. s6.child[0] -> d6.child[0]
3. s6.child[1] -> d6.child[1]
4. s6.child[2] -> d6.child[2]
5. s7 -> d7
6. s8 -> d8
7. s9 -> d9

【 0 0 6 0 】

次に、倍化型について説明する。

倍化型の変換の様子を図 1 0（C）に示す。図 1 3 は、図 1 0（C）の変換を、要素ごとの処理に分解して示した図である。

図 1 0（C）及び図 1 3 を参照すると、まず、モデル A のノード s 1 0 が、モデル B のノード d 1 0、d 1 4 に変換される。そして、モデル A におけるノード s 1 0 の子孫が、モデル B におけるノード d 1 0 及びノード d 1 4 の子孫にそれぞれ対応づけられる。これにより、次に、ノード s 1 0 の子孫であるノード s 1 1 がモデル B のノード d 1 1、d 1 5 に、ノード s 1 2 がモデル B のノード d 1 2、d 1 6 に、ノード s 1 3 がモデル B のノード d 1 3、d 1 7 に、それぞれ変換される。

以上の変換を実現する変換ルールを次に示す。

1. s10 -> d10, d14

2. s10.child[0] -> d10.child[0], d14.child[0]
3. s10.child[1] -> d10.child[1], d14.child[1]
4. s10.child[2] -> d10.child[2], d14.child[2]
5. s11 -> d11, d15
6. s12 -> d12, d16
7. s13 -> d13, d17

【 0 0 6 1 】

本実施の形態によれば、実施の形態 1 においてモデル A の変更が発生するたびに生成されるモデル B を格納するために消費されていた CPU 1 0 1 のキャッシュメモリやメインメモリ 1 0 3 の記憶領域を消費しなくて済む。したがって、モデルの変換コスト V_{conv} (式 (1) (2) 参照) において、メモリ空間使用量が $O(1)$ となり、計算量だけが $O(m)$ となり、メモリの使用効率が大幅に向上する。

【 0 0 6 2 】

〔実施の形態 2 の拡張〕

実施の形態 2 では、モデル変換をモデルの要素ごとに行うことによって、変換元モデル (モデル A) における変更のない部分の不要な変換を削減した。しかしながら、モデル A における変更に基づいて発生するアプリケーション A におけるイベントなどの情報に基づいて、変換先モデル (モデル B) における変更箇所が予測できるならば、モデル変換による新たなモデル B の生成及びモデル B における更新前後の差分抽出における処理効率を向上させることができる。

【 0 0 6 3 】

すなわち、変換元であるアプリケーション A 及び変換先であるアプリケーション B の種類 (すなわちモデル A 及びモデル B の種類)、及びモデル A の変更の内容によっては、モデル B における変更箇所が予測できる場合がある。

具体的には、モデル A の要素に変更が加わったときに、モデル変換の結果において、モデル B における対応要素以外に変更が加わらないことが条件となる。

【 0 0 6 4 】

このような予測ができる場合、モデル変換器 4 0 は、モデル A、B における全ての要素を副変換器 4 1 にて処理する必要はなく、変更箇所の要素のみを副変換器 4 1 にて処理すれば良いこととなる。

また、アプリケーション B 実行部 3 0 は、モデル B 全体における更新前後の差分を抽出する必要はなく、副変換器 4 1 によって処理された要素においてのみ更新前後の差分を抽出すれば良いこととなる。

【 0 0 6 5 】

また、モデル A の要素数に対して対応するモデル B の要素数が増減するような変換に対しても、本手法を適用して差分抽出の効率化を図ることができる。

本手法を適用する第 1 の方法は、モデル A、B 間で対応要素が存在しない場合、対応要素が存在する部分まで差分生成範囲を拡大し、当該拡大された差分生成範囲に対して変換処理を行う。ここで、差分生成範囲の拡大は、例えばツリー構造のモデル A、B の場合、対応要素が存在しないことが検出された要素からルート方向へ拡大する。

モデル A の要素数に対して対応するモデル B の要素数が増減するような変換に対して本手法を適用する第 2 の方法は、変換ルールを参照して差分生成範囲を決定する方法である。

【 0 0 6 6 】

図 1 0 (A) の減少型の変換において、モデル A におけるノード s 2 の child 属性に変更が加わった場合を例に考える。ここで、当該ノード s 2 自体の変換先の要素は存在しないが、第 1 の方法では、ノード s 1 以下の部分木を差分生成範囲とすることができる。また、第 2 の方法では、ノード s 2 の child 属性はノード d 1 の child 属性に変換されるというルールから、モデル生成、差分生成の対象ノードをノード d 1 と決定することができる。

【 0 0 6 7 】

図 1 4 は、この条件を満足しない、すなわち、モデル A における変更からモデル B における変更を予測できない形のモデル変換の例を示す図である。

図 1 4 を参照すると、まず、モデル A のノード s 1 4 が、モデル B のノード d 1 8 に変換される。そして、モデル A におけるノード s 1 4 の子孫が、モデル B

におけるノード d 1 8 の子孫に対応づけられる。これにより、次に、ノード s 1 4 の子孫であるノード s 1 5、s 1 6、s 1 7 が、それぞれモデル B のノード d 1 9、d 2 0、d 2 1 に変換される。

ここで、モデル A におけるノード s 1 5 の子孫が、モデル B の対応要素であるノード d 1 9 ではなく、ノード d 2 0 の子孫に対応づけられる。これによって、ノード s 1 5 の子孫であるノード s 1 8 が、モデル B のノード d 2 2 (ノード d 2 0 の子孫) に変換される。この部分の変換が上述した条件を満足しないため、図 1 4 のモデル変換では、モデル A の変更からモデル B の変更箇所を予測できないこととなる。

以上の変換を実現する変換ルールを次に示す。

1. s14 → d18
2. s14.child[0] → d18.child[0]
3. s14.child[1] → d18.child[1]
4. s14.child[2] → d18.child[2]
5. s15 → d19
6. s16 → d20
7. s17 → d21
8. s15.child[0] → d20.child[0]
9. s18 → d22

上記の変換ルールのうちで、8 番目のルールが上記の条件を満足しないため、図 1 4 のモデル変換ではモデル A の変更からモデル B の変更箇所を予測することができない。

【 0 0 6 8 】

しかしながら、図 1 4 のようなモデル変換が存在しなければ、モデル A の変更からモデル B の変更箇所を予測することが可能であるため、適用範囲を限定して本手法を適用することにより、モデル変換及びモデル B における変更前後の差分抽出において作業コストを大幅に低減することができる。

本手法による作業コストの低下を具体的に考察する。

まず、説明の簡単化のためにエレメント数が増加しない場合を考える。この場合、モデルの変換コスト V_{conv} 及び差分作成コスト V_{diff} の計算量は、エレメント1つのみを変換、比較すれば良いので $O(1)$ となる。したがって、イベント解釈及び再描画に要するコスト V_{update} のみが定数でない $O(n)$ となる。したがって、上記の式(2)で定義された、更新後のモデル B を反映したビュー B を改めて作り直す手法における作業コスト V_{t0} に比べて、大幅な作業コストの低減が見込まれる。

次に、エレメント数が増加した場合は、上述したように差分生成範囲を拡大しなければならないが、この範囲は変換ルールなどによって決まっている。したがって、モデルの変換コスト V_{conv} 及び差分作成コスト V_{diff} の計算量は、共に $O(1)$ となる。したがって、上記の式(2)で定義された作業コスト V_{t0} に比べて、大幅な作業コストの低減が見込まれる。

【0069】

〔実施の形態3〕

次に、変換先モデル（モデル B ）の更新前後の差分を抽出することなくモデル A の変更に基づいてビュー B を更新するイベントを発生させる手法について説明する。

アプリケーション A におけるモデル A を表示するためにアプリケーション B におけるビュー B を使用するためにモデル変換を行う状況で、アプリケーション A における編集作業をビュー B に反映させることは、変換元のモデル A が変更された場合に、変換先のビュー B に対してどのようなイベントを発生させれば良いかという問題に帰着できる。

上述した実施の形態1、2では、モデル A の変更に伴ってモデル変換によりモデル B を更新し、当該モデル B において更新前後の差分を抽出し、この差分に基づいてモデル B における変更内容を特定し、ビュー B を更新するためのイベントを発生させていた。

これに対し、アプリケーション A においてビュー A を更新するために生成されるイベントを、アプリケーション B においてビュー B を更新するためのイベント

に変換することにより、直接的にビュー B を更新するためのイベントを生成することもできる。

【 0 0 7 0 】

本実施の形態は、実施の形態 1 と同様に、図 1 に示すようなコンピュータ装置にて実現される。

図 1 5 は、本実施の形態によるアプリケーションの編集を行うための機能ブロックを示す図である。

図 1 5 を参照すると、本実施の形態は、アプリケーション A を実行するアプリケーション A 実行部 1 0 と、アプリケーション A におけるモデル A をアプリケーション B におけるモデル B に変換するモデル変換器 2 0 と、アプリケーション A においてビュー A を更新するイベントをアプリケーション B におけるビュー B を更新するイベントに変換するイベント変換器 5 0 と、アプリケーション B を実行するアプリケーション B 実行部 3 0 とを備える。

これらの構成要素のうち、アプリケーション A 実行部 1 0、モデル変換器 2 0 及びアプリケーション B 実行部 3 0 は、図 2 に示したアプリケーション A 実行部 1 0、モデル変換器 2 0 及びアプリケーション B 実行部 3 0 と同様であるため、同一の符号を付して説明を省略する。また、アプリケーション A とアプリケーション B との関係は、実施の形態 1 において図 3 を参照して説明したものと同様である。

なお、図 1 5 に示すイベント変換器 5 0 は、図 2 に示した各構成要素と同様に、図 1 のメインメモリ 1 0 3 にロードされたコンピュータプログラムにより制御された CPU 1 0 1 にて実現される仮想的なソフトウェアブロックである。

【 0 0 7 1 】

図 1 5 の構成において、イベント変換器 5 0 は、アプリケーション A でモデル A が更新された場合に、これに伴ってビュー A を更新するために生成されるイベントを入力する。そして、当該変換元のイベントとモデル変換器 2 0 がモデル A をモデル B に変換するために用いる変換ルールとに基づき、アプリケーション B のビュー B においてモデル A の変更を反映させた更新を行うためのイベントを生成する。

なお、本実施の形態では、イベント変換器 5 0 によってビュー B を更新するためのイベントが生成されるため、アプリケーション B 実行部 3 0 は、当該イベントに基づいてビュー B を更新すれば良い。したがって、図 2 に示した差分抽出部 3 1 及びイベント生成部 3 2 は不要となる。

【 0 0 7 2 】

図 1 6 は、上述したアプリケーション A 実行部 1 0、モデル変換器 2 0、イベント変換器 5 0、アプリケーション B 実行部 3 0 によるモデル及びビューの更新の動作を説明するフローチャートである。また、図 1 7 は、かかる更新動作を図 3 のモデル・ビュー対に対して適用した様子を示す図である。図 1 7 中に示された符号は、図 1 6 のステップに対応する。

図 1 6 に示すように、変換元のビュー A に書き込みが発生すると（ステップ 1 6 0 1）、アプリケーション A 実行部 1 0 により、変更されたビュー A に対応してモデル A が変更される（ステップ 1 6 0 2）。そして、アプリケーション A 実行部 1 0 からモデル変換器 2 0 へ、モデル A の更新が通知される。

次に、アプリケーション A 実行部 1 0 からの通知を受け取ったモデル変換器 2 0 が、モデル A における変更を反映させたモデル B を生成し、アプリケーション B 実行部 3 0 に渡す（ステップ 1 6 0 3）。

【 0 0 7 3 】

また、アプリケーション A 実行部 1 0 において、モデル A の変更をビュー A に反映させるためのイベントが生成される（ステップ 1 6 0 4）。そして、イベント変換器 5 0 が、アプリケーション A 実行部 1 0 により生成されたイベントとモデル変換ルールとを用いて、モデル B の変更をビュー B に反映させるためのイベントを生成し、アプリケーション B 実行部 3 0 へ通知する（ステップ 1 6 0 5）。そして、アプリケーション B 実行部 3 0 において、このイベントによって、ビュー B が、モデル A の変更内容を反映させた表示に更新される（ステップ 1 6 0 6）。

【 0 0 7 4 】

ところで、モデル A、B がツリー構造のモデルである場合、一般的な変換ルールは、1 対 1 でノードが対応する変換（通常）、ノードの削除、ノードの挿入及

びノードに付加された属性の変更（入れ替え）の種類であり、複雑な変更であってもこれらの組合せで表現できる（この他、特殊な場合として、実施の形態2において説明した図10に示すノード（要素）の個数が増減する変換がある）。

図18は、これらの変換の様子を示す図である。

これらのうち、ツリー構造が変化する3種類の変換ルール、すなわちノードの削除、ノードの挿入及びノードに付加された属性の変更に関して、モデルAの該当要素に適用される変換ルールに基づき、モデルBに対してどのようなイベントを発生させるかについて説明する。

【0075】

変換元であるモデルAの該当要素（以下、変換元要素）をdとし、変換先であるモデルBにおける対応要素（以下、変換先要素）を得るためのマッピングをmapとし、要素dから得られる変換先要素をmap(d)とする。変換元、変換先の各要素は、それぞれ、親を示す「parent」、子供を示す配列型の「children」、自分が親に対して何番目の子供かを示す「index」という属性を持つと仮定する。また、モデルAで発生するイベント（以下、変換元イベント）をse、モデルBで発生させるイベント（以下、変換先イベント）をdeとし、それぞれ、変更対象のノードを示す「target」、挿入・削除の対象を示す「element」、挿入・削除の行われた個所を示す「index」、イベントの種類を示す「type (REMOVE、INSERT、CHANGEDのいずれか)」という4つの属性を持っているものとする。

【0076】

以上のような条件で、イベント変換器50は、変換元イベントseからマッピングmapを用いて、次のようにして変換先イベントdeの属性を決定する。

・ de.type = f(se.type, se.target, se.element) ただし、fは、変換元要素dにおける変更の種類、変更対象、挿入・削除対象から変換先の変更の種類を得る関数。

・ de.target = map(se.target) ただし、この値がNullの場合は、変換先イベントdeを破棄する（イベントを生成しない）。

・ de.element = map(se.element)

・ de.index = map(se.element).index

【 0 0 7 7 】

以下、上述した実施の形態を具体的なアプリケーションに適用した例について説明する。

〔適用例 1〕

本適用例では、アプリケーションとして、HTMLのDOMを、米国サン・マイクロシステムズ社より提供されているJava 2 Platform, Standard Edition付属のテキストコンポーネントにマッピングする方法で作られたHTMLエディタを挙げる。

図 1 9 は、このHTMLエディタの内部構造を示す図である。

図 1 9 において、モデルHTMLDOMは、HTMLにおけるDOMのインターフェイスを持ったツリー構造のモデルである。

また、ビューJTree(0) (ツリー表示) は、javax.swing.JTreeのインスタンスである。モデルTreeNodeは、ビューJTree(0)のモデルであり、DOMから変換されたjavax.swing.tree.TreeNodeインターフェイスを実装している。

また、ビューJEditorPane (WYSIWYG表示) は、javax.swing.JEditorPaneのインスタンスである。モデルSwingDocumentは、ビューJEditorPaneのモデルであり、DOMから変換されたjavax.swing.text.Documentインターフェイスを実装する。さらに、このモデルSwingDocumentは、TreeNodeインターフェイスも実装しており、これに応じてツリー表示のビューJTree(1)を持っている。

【 0 0 7 8 】

ここで、モデルHTMLDOMとモデルTreeNodeとの間の変換は単純な1対1の変換であり、したがって、ビューJTree(0)は、モデルHTMLDOMのビューに等しい。これに伴って、以下の説明では、モデルHTMLDOMからモデルSwingDocumentへの変換を行う場合について説明する。また、モデル変換の手法は、上述した実施の形態2の手法 (すなわち、モデルにおける対応要素ごとに変換を行う手法) とする。

【 0 0 7 9 】

図 2 0 は、モデルHTMLDOMとしてIBM社のウェブページ (<http://www.ibm.com/>) を用い、ビューJTree(0)で表示した画像イメージを示す図である。図 2 1 は

、同ウェブページをビューJTree(1)で表示した画像イメージを示す図である。また図22は、同ウェブページをビューJEditorPaneで表示した画像イメージを示す図である。

【0080】

ここで、図22において中央付近に表示された、「Power packed offer」という題を含むイメージデータを示すDOMノード(IMG0)を削除する動作を行うものとする。

図23は、ノード(IMG0)を削除する前のモデルHTMLDOMにおける当該ノード(IMG0)付近の状態を示す図である。また、図24は、このモデルHTMLDOMから変換されたモデルSwingDocumentにおける対応要素であるノード(IMG1)付近の状態を示す図である。なお、図23において、モデルHTMLDOMのイメージは、ツリー表示を行うビューJTree(0)のイメージを用いて表現している。同様に、図24において、モデルSwingDocumentのイメージは、ツリー表示を行うビューJTree(0)を用いて表現している。

図23において、ノード(IMG0)の親ノードはノード(Anchor0)であり、図24において、ノード(IMG1)の親ノードはノード(Anchor1)である。

【0081】

モデルHTMLDOMからモデルSwingDocumentへ変換する変換ルールは、次のようになっている。

1. TABLE → DomTableElement
2. TABLE.child[0] → Null
3. TBODY → Null
4. TBODY.child[0] → DomTableElement.child[0], TBODY.child[1] → DomTableElement.child[1], ...
5. TR → DomTableRowElement
6. TD → DomTableCellElement
7. A → DomCharacterElement, DomCharacterElement.lastChild = EmptyElement (左項は最後の子供を示す)

8. IMG → DomSpecialElement

【 0 0 8 2 】

図 2 5 は、図 2 3 からノード (IMG0) を削除した状態を示す図である。また、図 2 6 は、このモデル HTMLDOM から変換されたモデル SwingDocument における対応要素であるノード (Anchor1) 付近の状態を示す図である。さらに、図 2 7 は、図 2 6 に示すモデル SwingDocument に対するビュー JEditorPane による画像イメージを示す図である。

図 2 5、図 2 6 を参照すると、モデル HTMLDOM においてノード (IMG0) が削除されたことに伴い、モデル SwingDocument においてもノード (IMG1) が削除されている。さらに、図 2 7 を参照すると、ビュー JEditorPane において、ノード (IMG1) に対応するイメージデータが削除されていることがわかる。

【 0 0 8 3 】

以上の例では、ノード (IMG0) が削除される変更が発生した変換元モデル (HTMLDOM) の要素がノード (Anchor0) であった。そこで、モデル変換器 4 0 は、副変換器 4 1 を用いて、モデル HTMLDOM のノード (Anchor0) をモデル SwingDocument のノード (Anchor1) に変換する。

次に、アプリケーション B 実行部 3 0 の差分抽出部 3 1 が、当該変換前に保持していたノード (Anchor1) とモデル変換器 4 0 から受け取ったノード (Anchor1) とを比較する。この比較によって、ノード (Anchor1) の第 1 子であるノード (IMG1) が削除されていることがわかり、これが差分として抽出される。

次に、アプリケーション B 実行部 3 0 のイベント生成部 3 2 が、この差分を反映させるためのイベントを生成し、ビュー JEditorPane に投げる。このイベントは、javax.swing.event.DocumentEvent インターフェイスを実装するインスタンスであり、実際の内容は次のようになっている。

DocumentEvent

- ・ イベント発生個所の文書先頭からのオフセット: 405
- ・ イベント対象領域の長さ: 1

- ・ イベントの種類: REMOVE
- ・ イベント発生エレメント: Anchor1
- ・ 変更後に保持している子供のエレメント: 0th child: EmptyElement [405, 406, name:content]
- ・ イベントの対象となる子供のインデックス: 0
- ・ 取り除かれた子供のエレメント: IMG0

このイベントをビューJEditorPaneが受け取ると、表示内容が図 2 7 に示したように更新される。

【 0 0 8 4 】

〔適用例 2〕

本適用例では、X S Lによって変換されたHTMLのプレビュー機能を持つXMLエディタを挙げる。なお、本適用例においても、モデル変換は実施の形態 2 の手法によるものとする。

図 2 8 は、このXMLエディタの内部構造を示す図である。

図 2 8 において、モデルTreeNode及びビューTreeEditorからなるモデル・ビュー対は、通常のXMLのツリーエディタ（例えば、I B M社のXeena）である。ツリーエディタのモデルTreeNodeは、モデル変換器 4 0 であるX S L Tプロセッサによって、HTMLのDOM（モデルHTMLDOM）に変換される。また、モデルHTMLDOMは、ビューHTML ViewerにてW Y S I W Y G表示される。ビューHTML Viewerには、一般的なウェブブラウザを用いることができる。

【 0 0 8 5 】

ツリーエディタによって、モデルTreeNodeに変更が加えられた場合、X S L Tプロセッサに変更箇所が通知され、X S L Tプロセッサによる変換を経て、モデルHTMLDOMにおける対応箇所が変更される。

次に、アプリケーションB 実行部 3 0 によって、当該変更前後のモデルHTMLDOMの差分が計算される。そして、この差分をパラメータとして、ビューHTML Viewerを更新するためのイベントが生成され、ビューHTML Viewerに投げられる。ビューHTML Viewerは、受け取ったイベントに基づいて、モデルTreeNodeの変更を

反映させるように更新される。

【 0 0 8 6 】

上記のXMLエディタを用いた動作例として、図29に示す文書型のXMLデータを、携帯情報端末などで用いられるコンパクトHTMLに変換してプレビューする例を考える。

図30は、この変換を実現するXSLスタイルシートの例を示す図である。

また、図31は、変換元モデル（モデルA）であるXMLデータの例を示す図、図32は、図31のモデルを変換して得られる変換先モデル（モデルB）であるコンパクトHTMLデータを示す図である。ただし、図31、図32は、それぞれソース表示のビューA、Bにて表示された状態が示されている。

【 0 0 8 7 】

ここで、変換元モデルであるXMLデータ（図3.1）に対する変更として、contentエレメントの次に、以下の文書の断片で示された部分木を挿入する。

```
<statement subject="OK">
  <content>OK</content>
</statement>
```

この変更に応じて、モデル変換器40であるXSLプロセッサが、XMLデータ（図31）中のstatementエレメントに関して変換を行い、当該変更を反映させたコンパクトHTMLデータを生成する。図33は、生成されたコンパクトHTMLデータを示す図である。

【 0 0 8 8 】

この変換の際に、1番目のulエレメント以下の差分計算がなされ、2番目のulエレメントにliエレメントが付け加えられたことがわかる。そこで、アプリケーションB実行部30が、次に示すようなDOMの更新イベントを生成し、ビューHTML Viewerを更新する。

MutationEvent

イベントの種類：INSERT

イベント発生エレメント：2番目のulエレメント

イベントの対象となる子供のインデックス：0

付け加わった子供のエレメント：2番目のliエレメント

【0089】

〔適用例3〕

以上の適用例では、モデルがツリー構造である場合について説明したが、本実施の形態は、ツリー構造以外の構造を持ったモデルの変換に対しても応用可能である。本適用例では、変換元モデル（モデルA）が2次元配列であり、これをHTMLのテーブルに変換する例を挙げる。

図34は、モデルAに変更を加える前のモデルA、Bを示す図である。

図34において、モデルAは表形式によるビューで示し、モデルBはHTMLのテーブルをツリー表示によるビューで示している。

【0090】

図34に示したモデルAにおいて、当期利益の列を削除する変更を行う場合を考える。

図35は、当該変更が行われた場合のモデルA、Bを示す図である。

図35のモデルAに示すように、当期利益の列が削除されている。かかる変更がモデルAに行われると、モデル変換によって変更内容がモデルBに反映される。ここで、モデルBにおけるTABLEエレメント以下の変更前後の差分を作成すると、それぞれのTRエレメントから、最も右に位置していたエレメント（TH/TDエレメント）が削除されたことがわかる。これにより、次に示すイベントが、それぞれのTRエレメントに対して発生する。

MutationEvent

イベントの種類：REMOVE

イベント発生エレメント：TRエレメント

イベントの対象となる子供のインデックス：3

取り除かれた子供のエレメント：TD(最初のTRに対してはTH)エレメント

以上の4つのイベントは、同時に発生するものであるが、一般にはビューとモデルと間でイベントの並列な通知はできないので、直列化して順番にビューに対して通知する。

【0091】

【発明の効果】

以上説明したように、本発明によれば、モデル変換を用いて所定のモデルから他のアプリケーションのビューを使用する場合に、変換元のモデルに対する変更を即時的に変換先のモデル及びビューに反映させ、ビューを更新することが可能となる。

【0092】

また、本発明によれば、モデル変換を用いて所定のモデルから他のアプリケーションのビューを使用する場合に、変換元のモデルの変更に対応する部分的な変更を変換先のビューに対して行うことによりビューを更新することが可能となる。

【図面の簡単な説明】

【図1】 実施の形態1によるアプリケーション編集装置の実現に好適なコンピュータ装置の構成例を模式的に示した図である。

【図2】 本実施の形態によるアプリケーションの編集を行うための機能ブロックを示す図である。

【図3】 本実施の形態における変換元のアプリケーションと変換先のアプリケーションとの関係を説明する図である。

【図4】 本実施の形態におけるモデル及びビューの更新の動作を説明するフローチャートである。

【図5】 図4に示す更新動作を図3のモデル・ビュー対に対して適用した様子を示す図である。

【図6】 実施の形態2によるアプリケーションの編集を行うための機能ブロックを示す図である。

【図 7】 本実施の形態におけるモデル変換器による変換処理の様子を示す図である。

【図 8】 本実施の形態における副変換器による変換処理の様子を示す図である。

【図 9】 本実施の形態におけるモデル変換器が副変換器による変換処理を統合してモデル変換を行う処理を説明するフローチャートである。

【図 1 0】 変換前後で要素の数が増減するモデル変換を説明する図である。

【図 1 1】 図 1 0 (A) の変換を要素ごとの処理に分解して示した図である。

【図 1 2】 図 1 0 (B) の変換を要素ごとの処理に分解して示した図である。

【図 1 3】 図 1 0 (C) の変換を要素ごとの処理に分解して示した図である。

【図 1 4】 モデル A における変更からモデル B における変更を予測できない形のモデル変換の例を示す図である。

【図 1 5】 実施の形態 3 によるアプリケーションの編集を行うための機能ブロックを示す図である。

【図 1 6】 本実施の形態におけるモデル及びビューの更新の動作を説明するフローチャートである。

【図 1 7】 図 1 6 に示す更新動作を図 3 のモデル・ビュー対に対して適用した様子を示す図である。

【図 1 8】 一般的な変換ルールによるモデル変換を説明する図である。

【図 1 9】 適用例 1 における HTML エディタの内部構造を示す図である。

【図 2 0】 本適用例におけるビュー JTree(0) の表示例を示す図である。

【図 2 1】 本適用例におけるビュー Jtree(1) の表示例を示す図である。

【図 2 2】 本適用例におけるビュー JEditorPane の表示例を示す図である。

【図 2 3】 ノード (IMG0) を削除する前のモデルHTMLDOMにおけるノード (IMG0) 付近の状態を示す図である。

【図 2 4】 図 2 3 のモデルHTMLDOMから変換されたモデルSwingDocumentにおける対応要素であるノード (IMG1) 付近の状態を示す図である。

【図 2 5】 図 2 3 からノード (IMG0) を削除した状態を示す図である。

【図 2 6】 図 2 5 のモデルHTMLDOMから変換されたモデルSwingDocumentを示す図である。

【図 2 7】 図 2 6 に示すモデルSwingDocumentに対するビューJEditorPaneによる画像イメージを示す図である。

【図 2 8】 適用例 2 におけるXMLエディタの内部構造を示す図である。

【図 2 9】 XMLデータの例を示す図である。

【図 3 0】 モデル変換を行うXSLスタイルシートの例を示す図である。

【図 3 1】 本適用例における変換元モデル (モデルA) であるXMLデータの例を示す図である。

【図 3 2】 図 3 1 のモデルを変換して得られる変換先モデル (モデルB) であるコンパクトHTMLデータを示す図である。

【図 3 3】 図 3 1 のモデルに対する変更を反映させたコンパクトHTMLデータを示す図である。

【図 3 4】 適用例 3 において、モデルAに変更を加える前のモデルA、Bを示す図である。

【図 3 5】 図 3 4 のモデルAに対して変更を加えた後のモデルA、Bを示す図である。

【図 3 6】 従来技術における基本的なモデルとビューとの分割を示す図である。

【図 3 7】 図 3 6 のアプリケーションにおいて、所定のビューにおける変更がモデル及び他のビューに反映する様子を示す図である。

【図 3 8】 Adapterパターンを模式的に例示する図である。

【図 3 9】 モデル変換により他のアプリケーションのビューを用いて表示を行う方法を説明する図である。

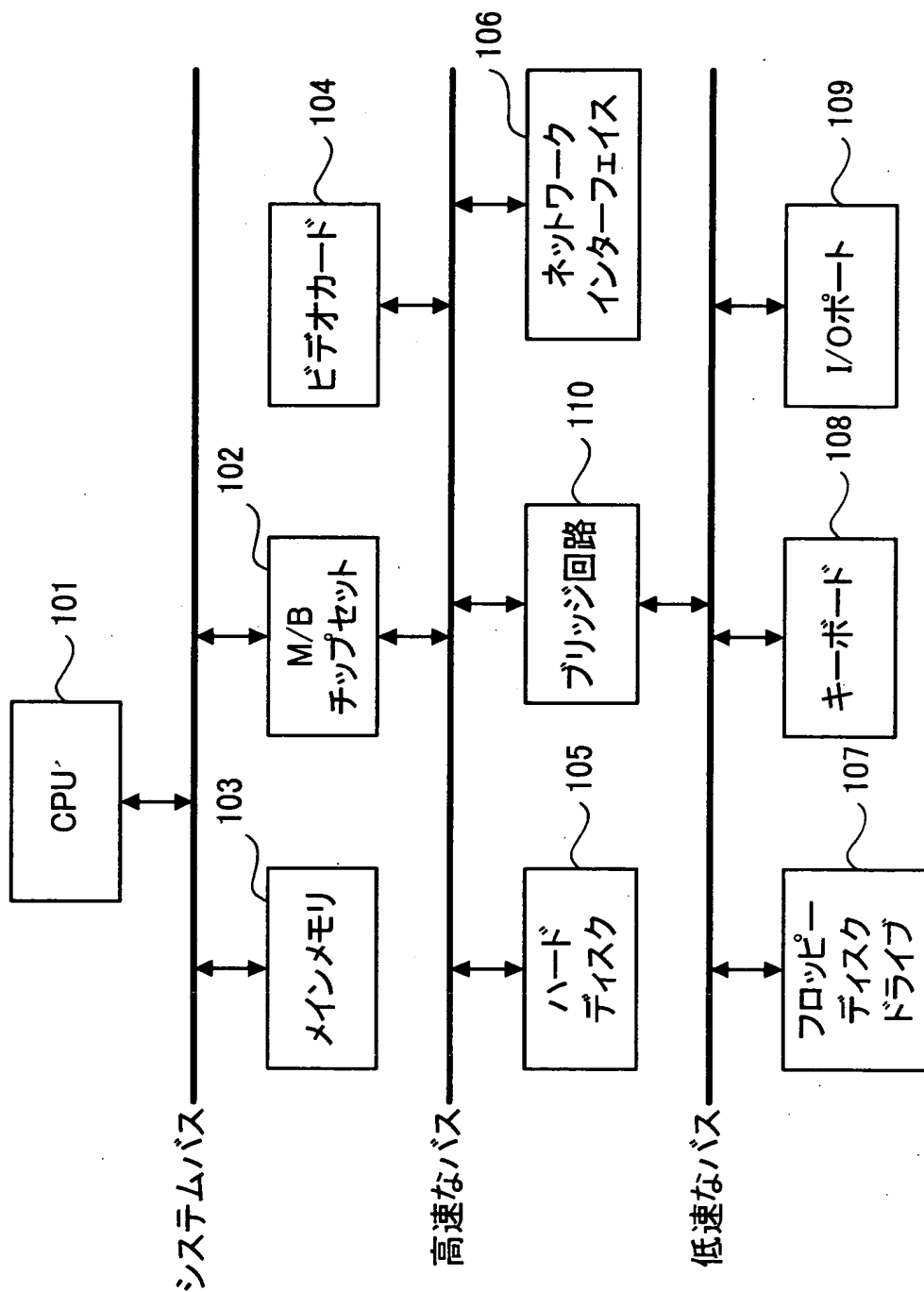
【符号の説明】

1 0 …アプリケーション A 実行部、2 0、4 0 …モデル変換器、3 0 …アプリケーション B 実行部、3 1 …差分抽出部、3 2 …イベント生成部、4 1 …副変換器、5 0 …イベント変換器

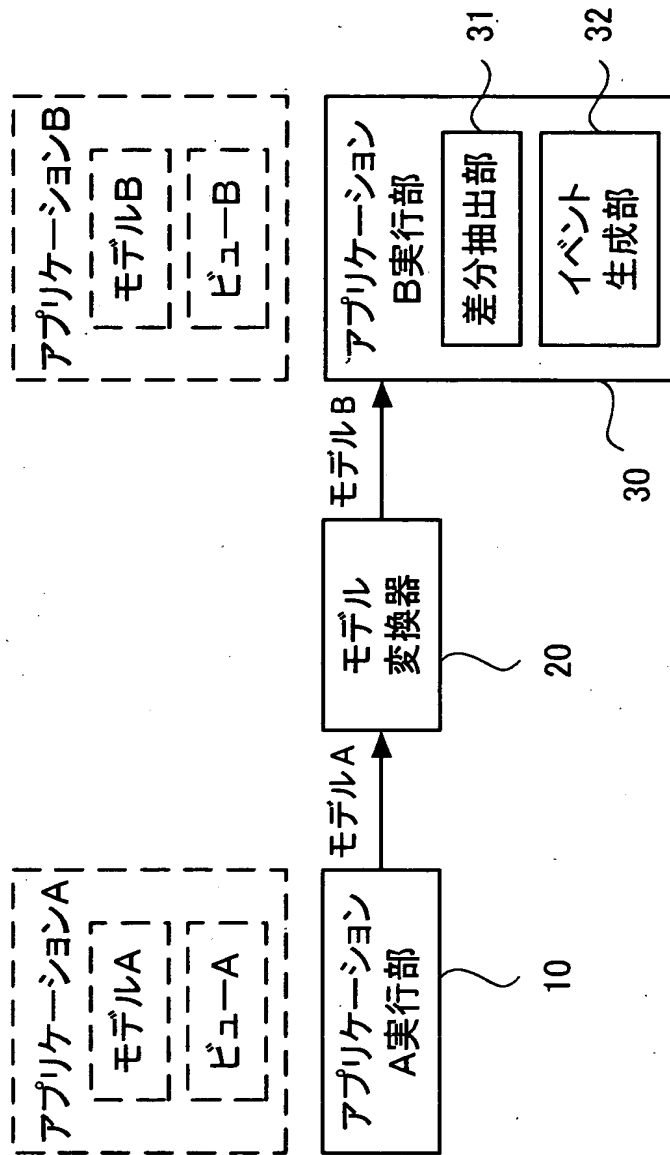
【書類名】

図面

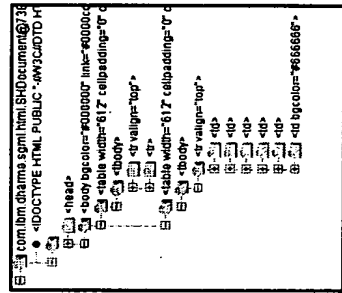
【図 1】



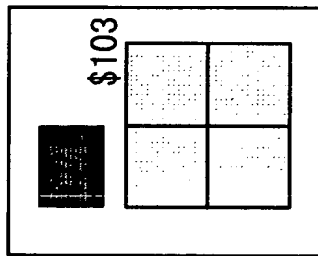
【図 2】



【図3】

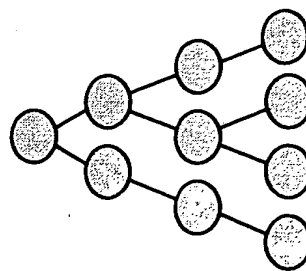


ビューB



ビューA

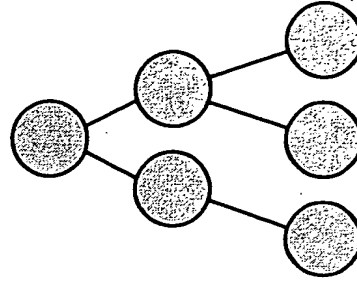
ビュー



モデルA

アプリケーションA

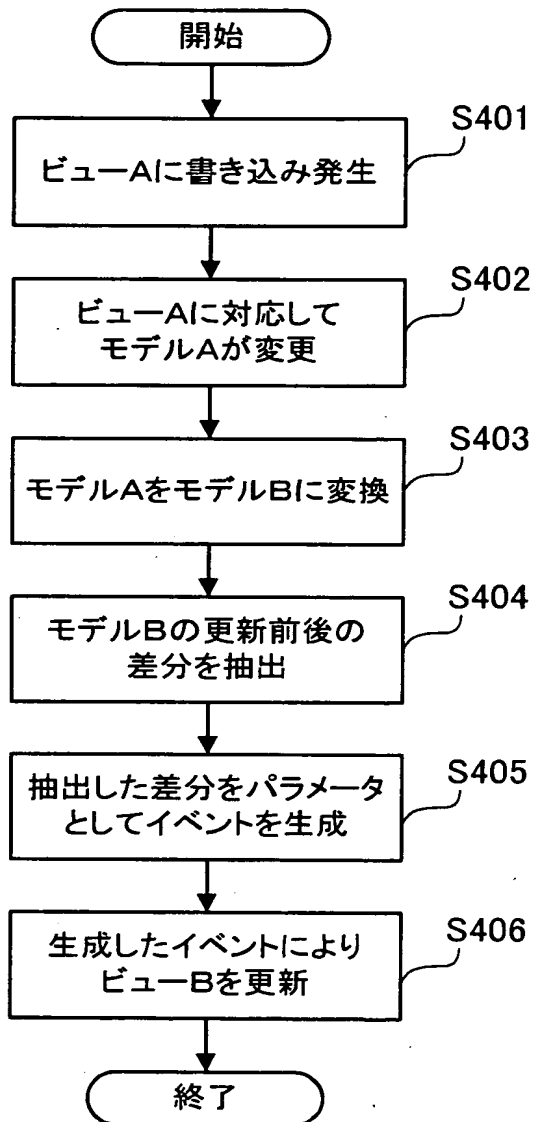
モデル変換



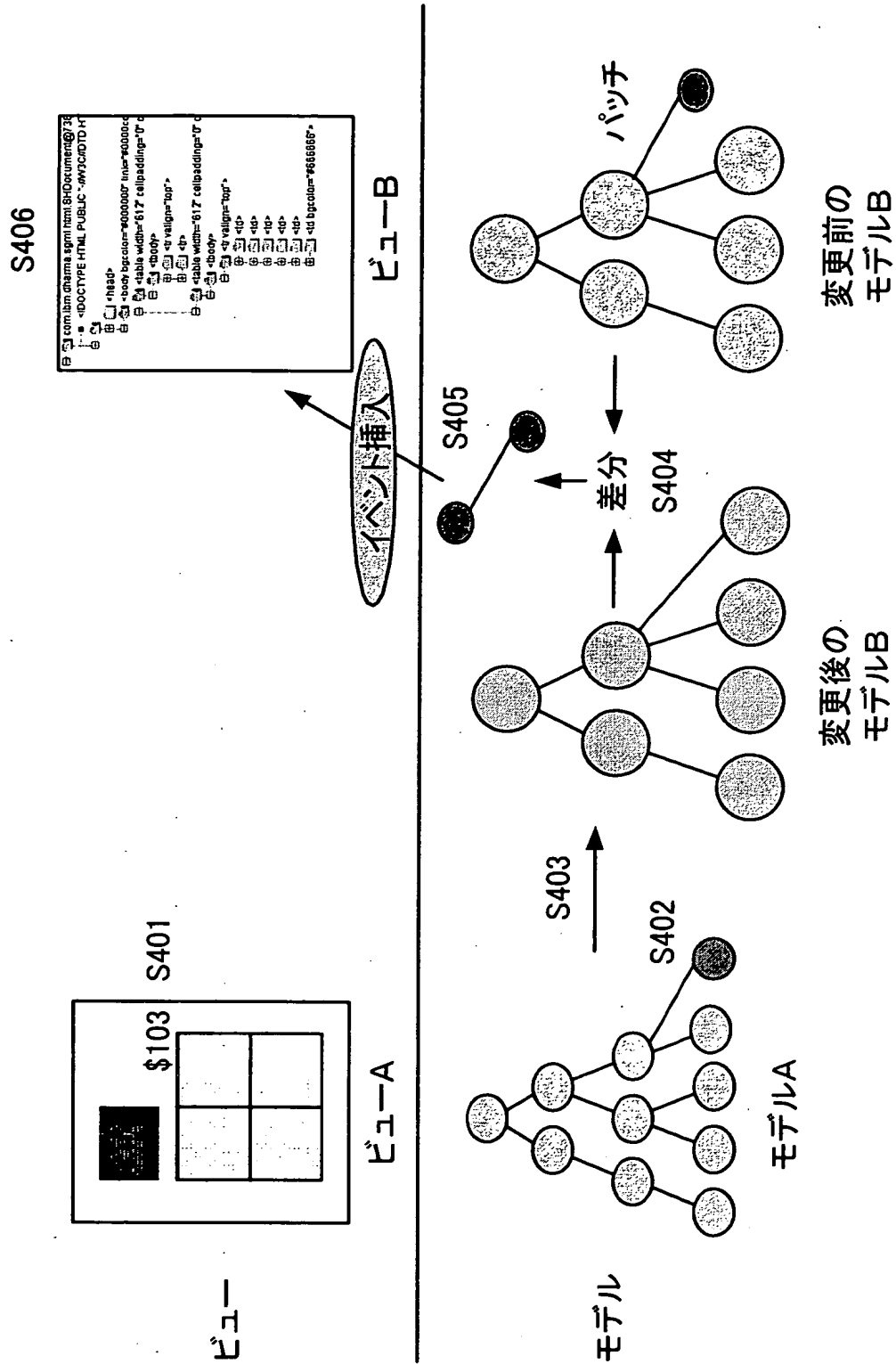
モデルB

アプリケーションB

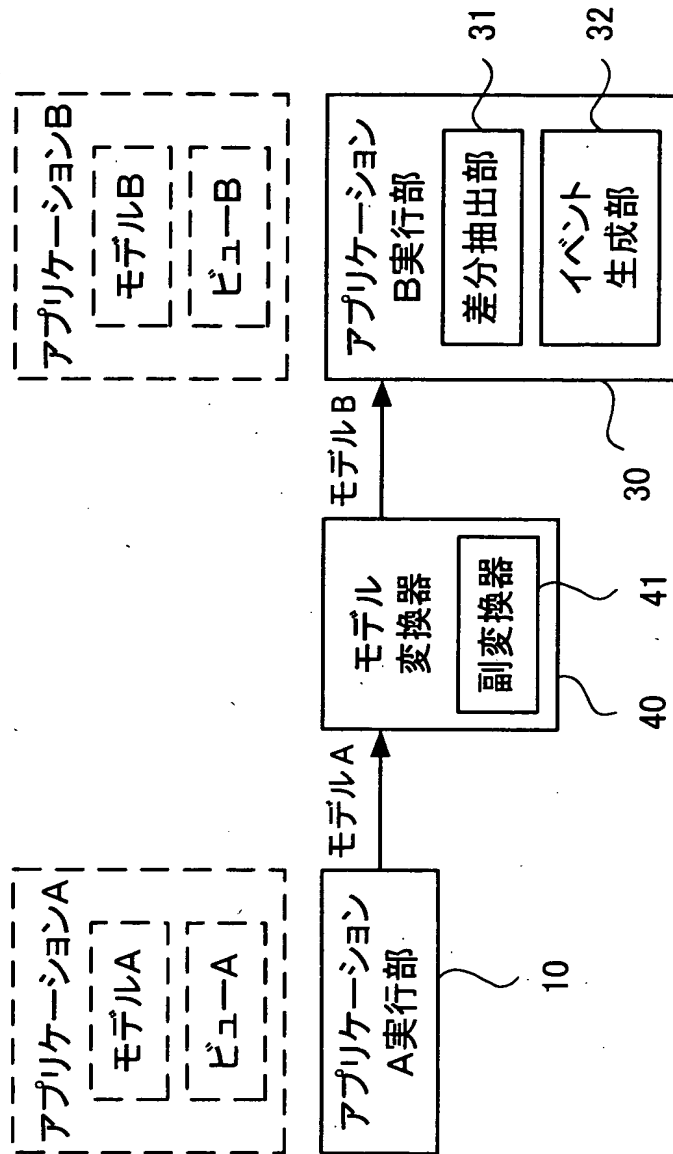
【図 4】



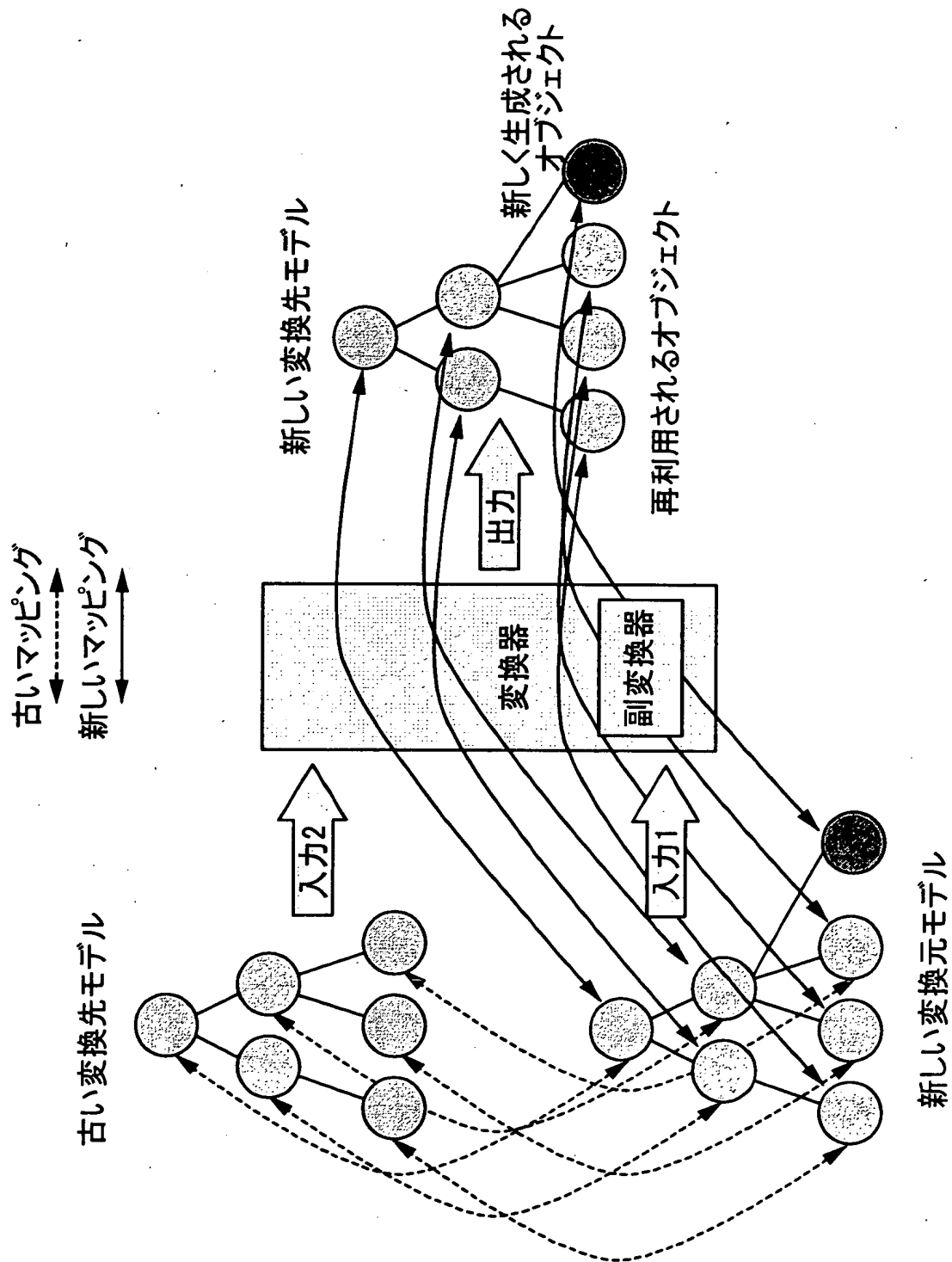
【図 5】



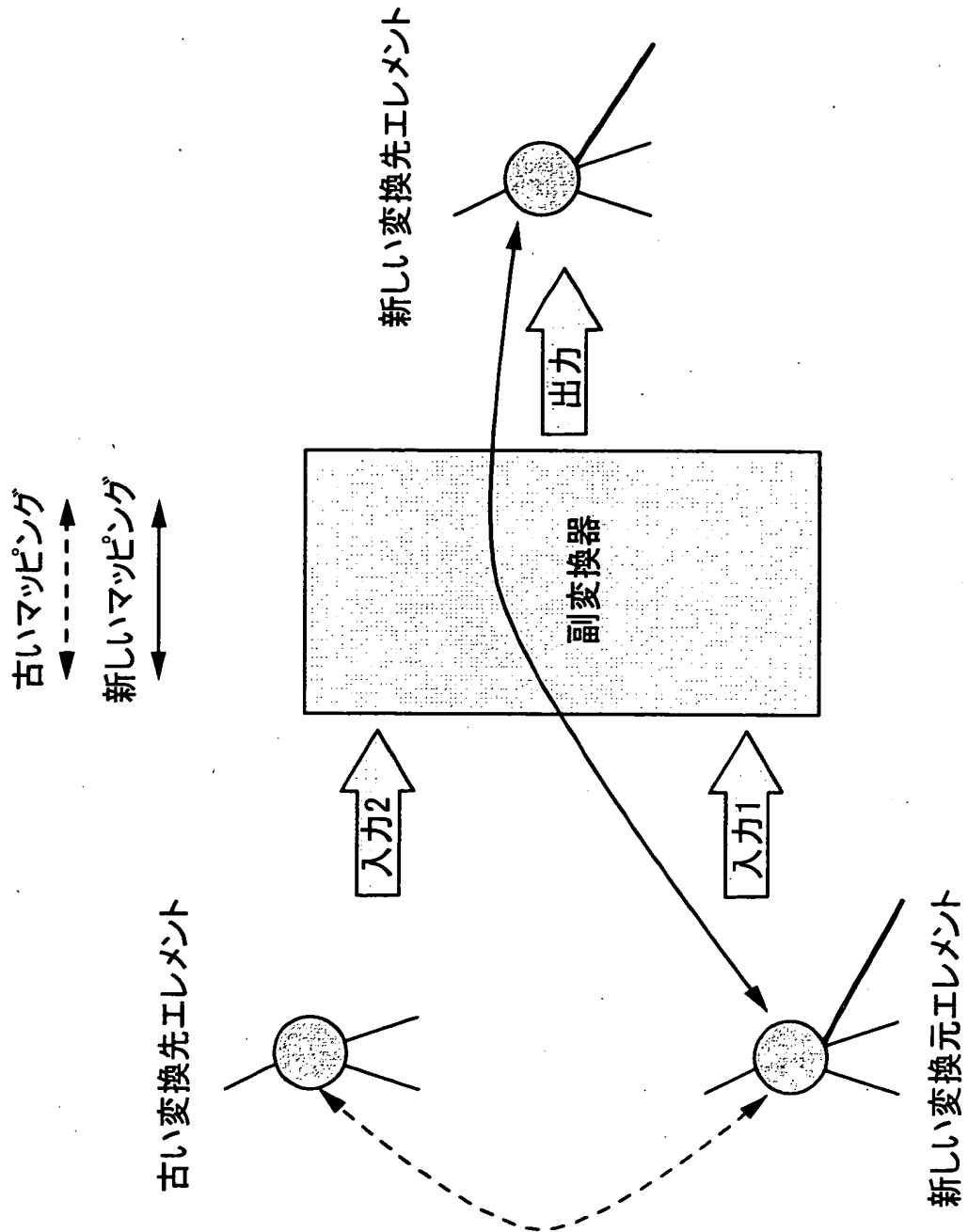
【図 6】



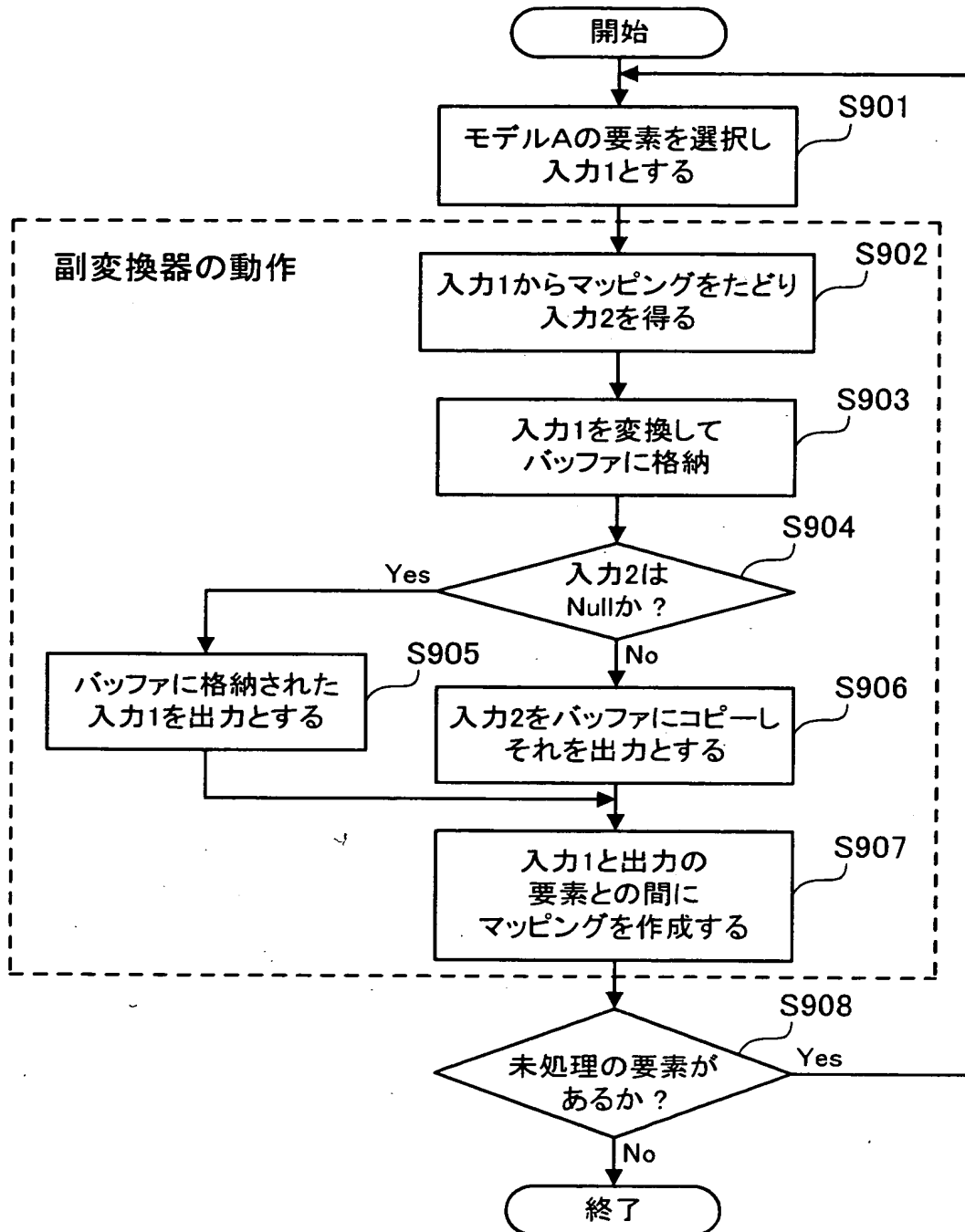
【図 7】



【図 8】

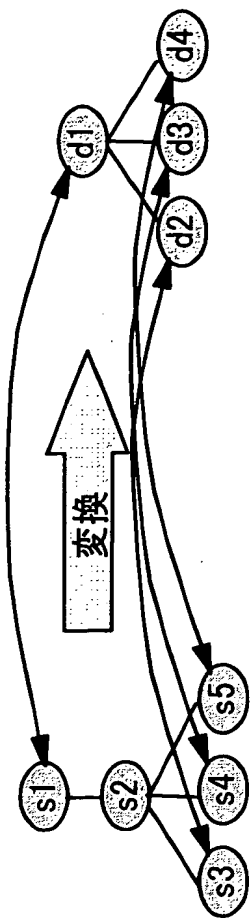


【図9】

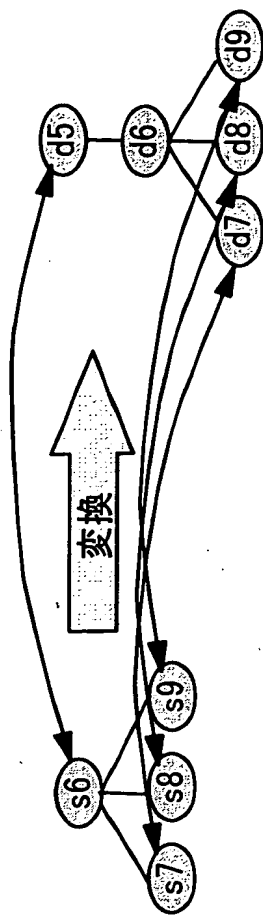


【図 1 0】

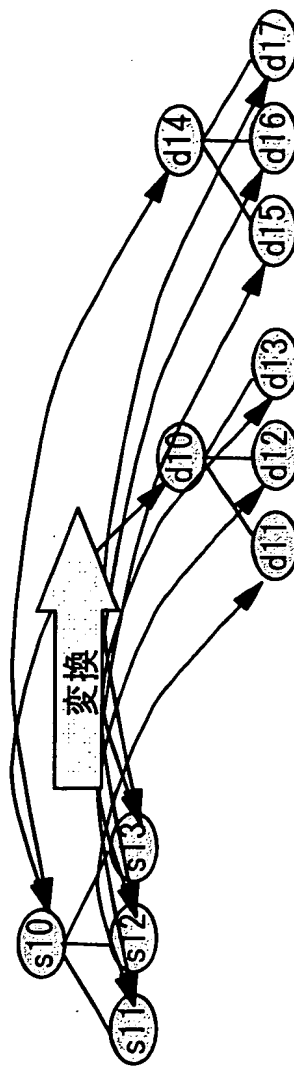
(A) 減少型



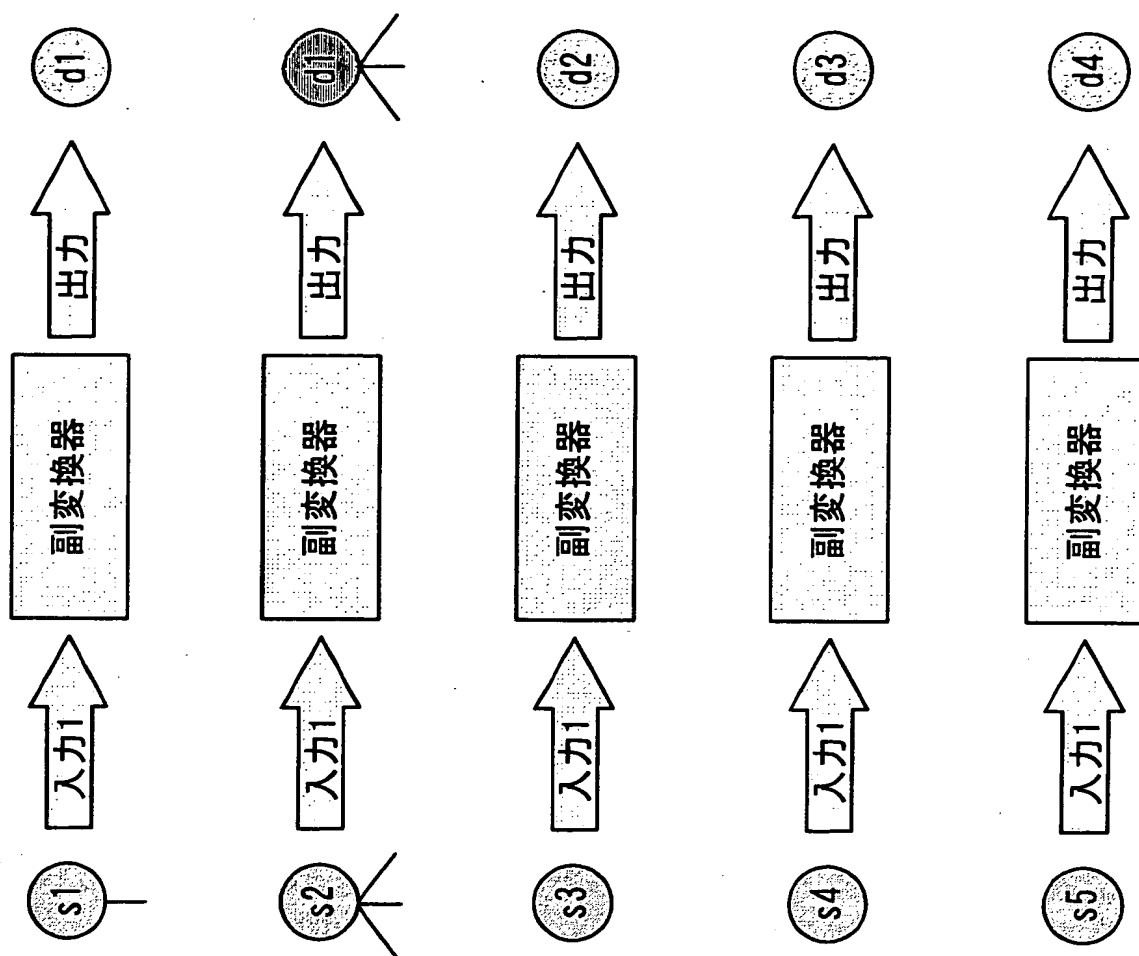
(B) 増加型



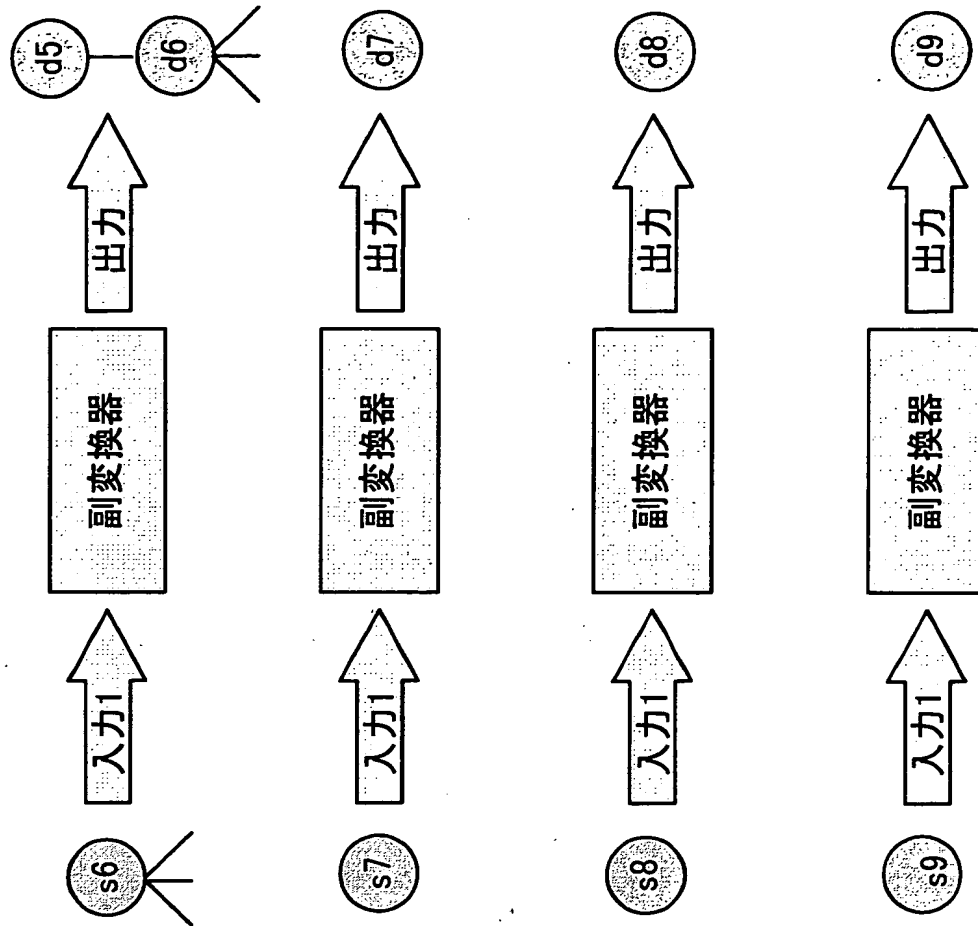
(C) 倍化型



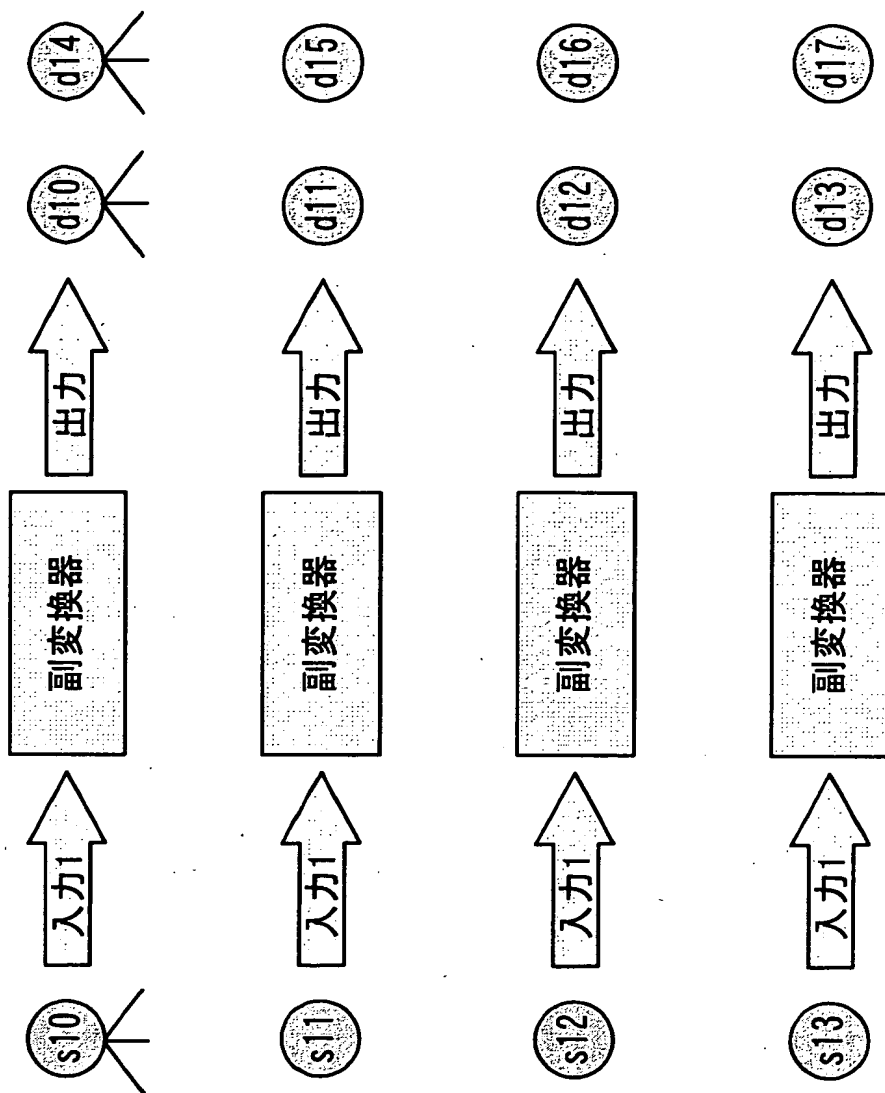
【図 1 1】



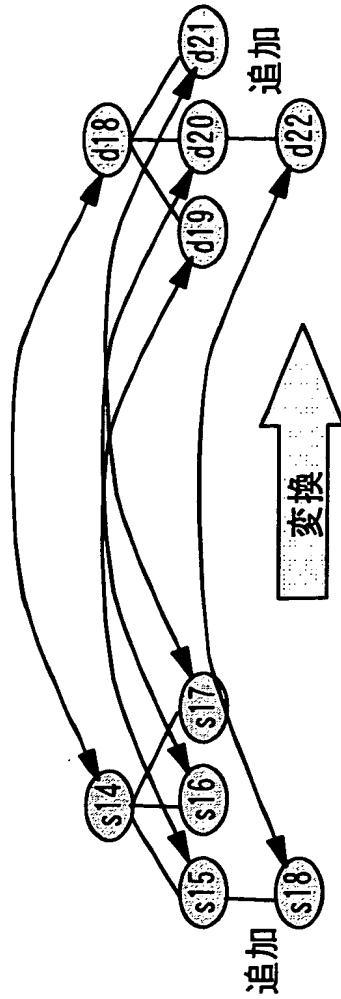
【図 1 2】



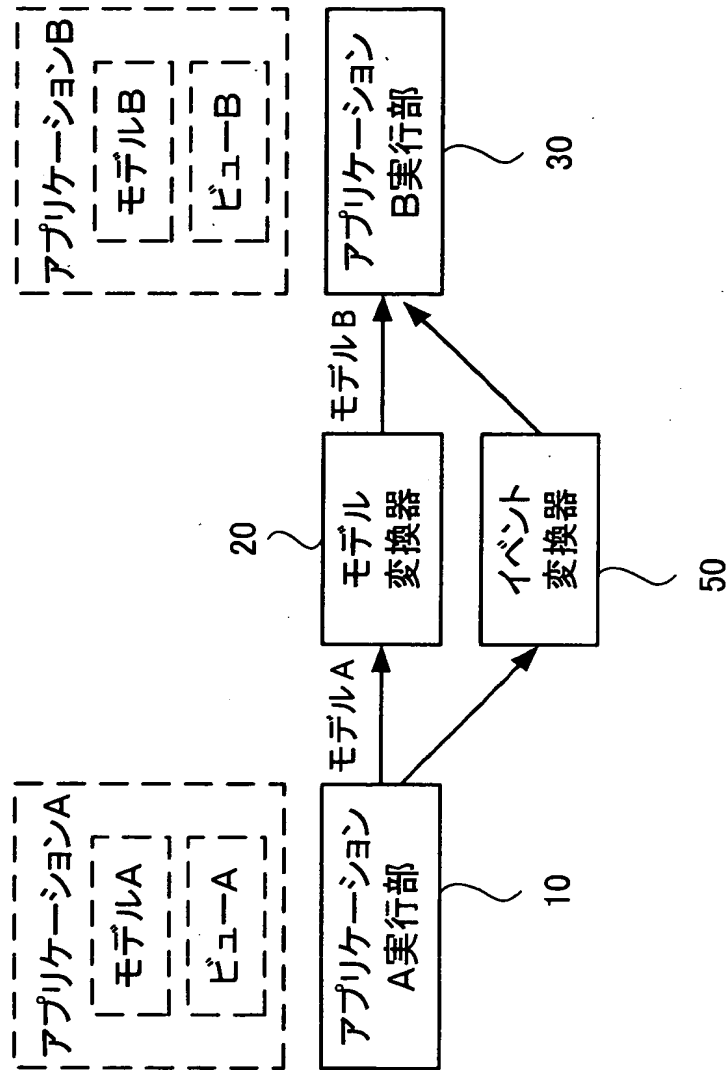
【図 1 3】



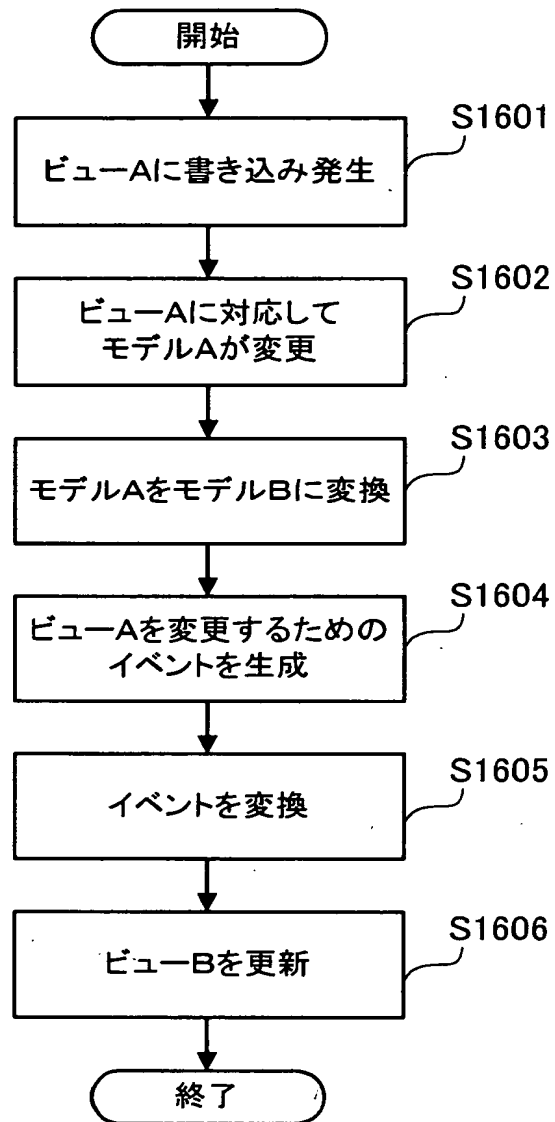
【図 1 4】



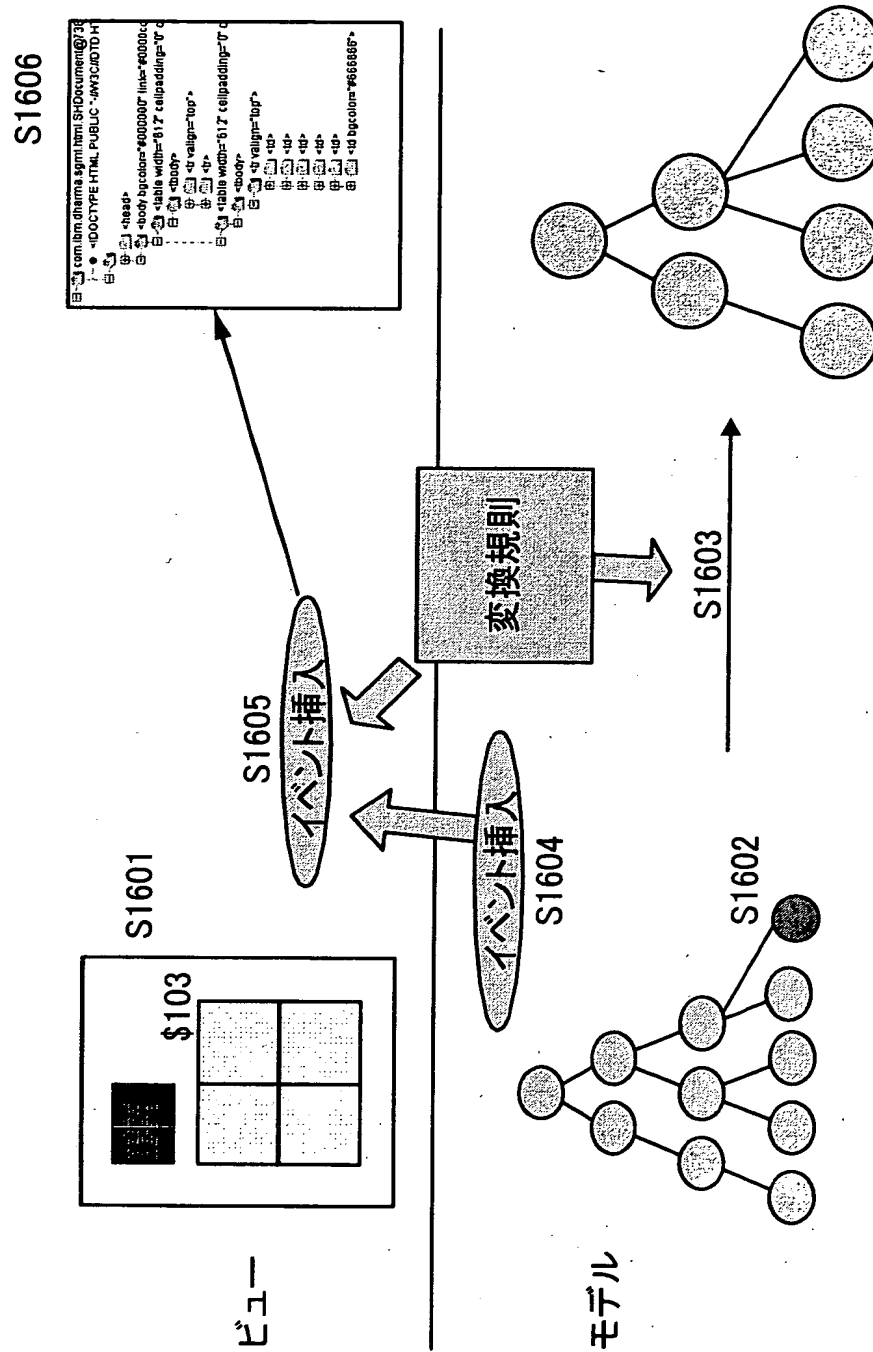
【図15】



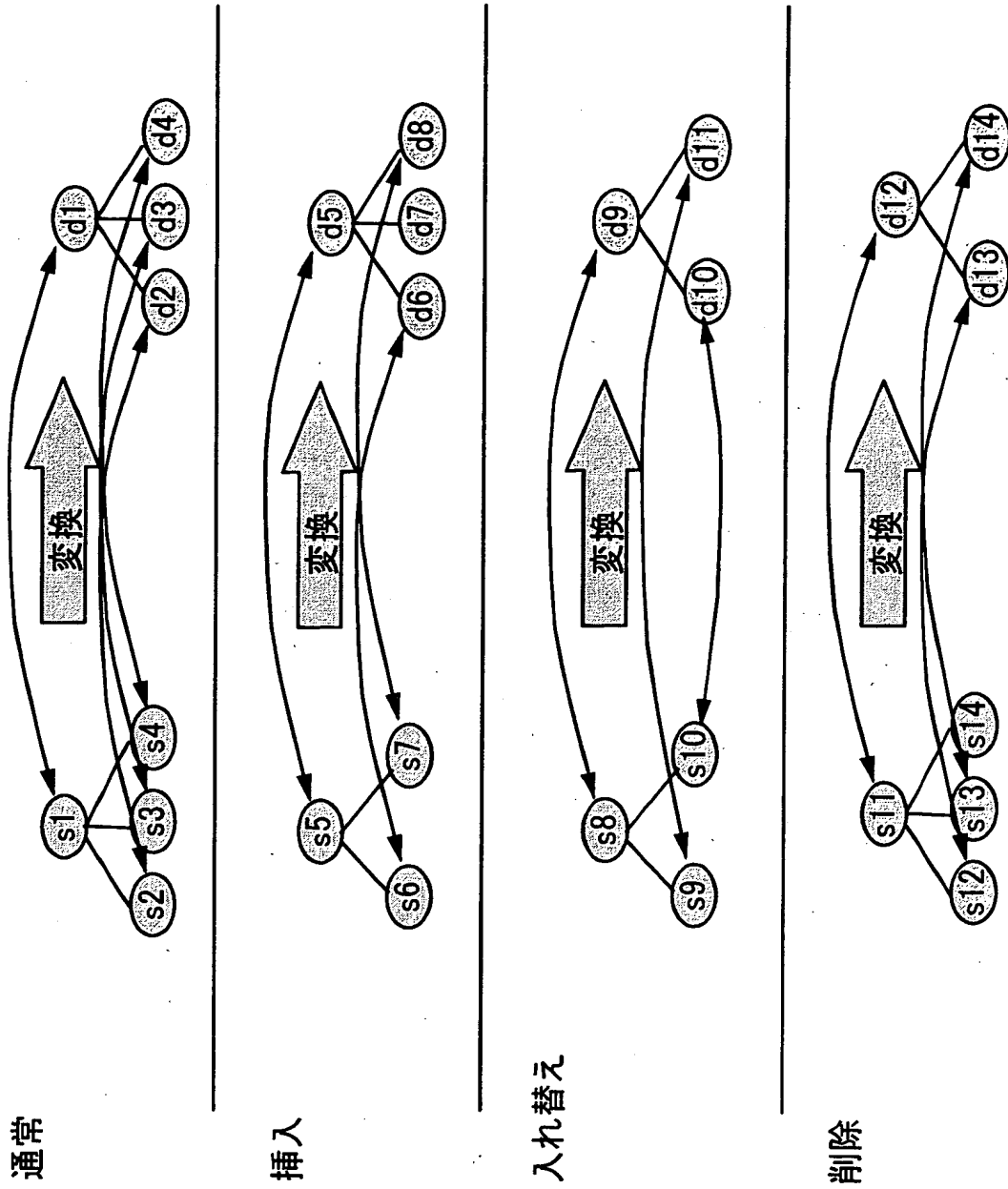
【図 1 6】



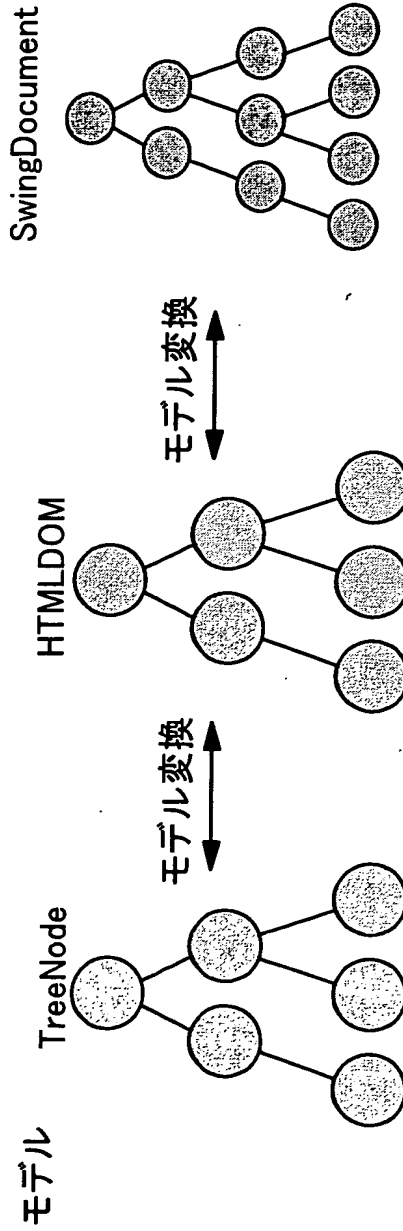
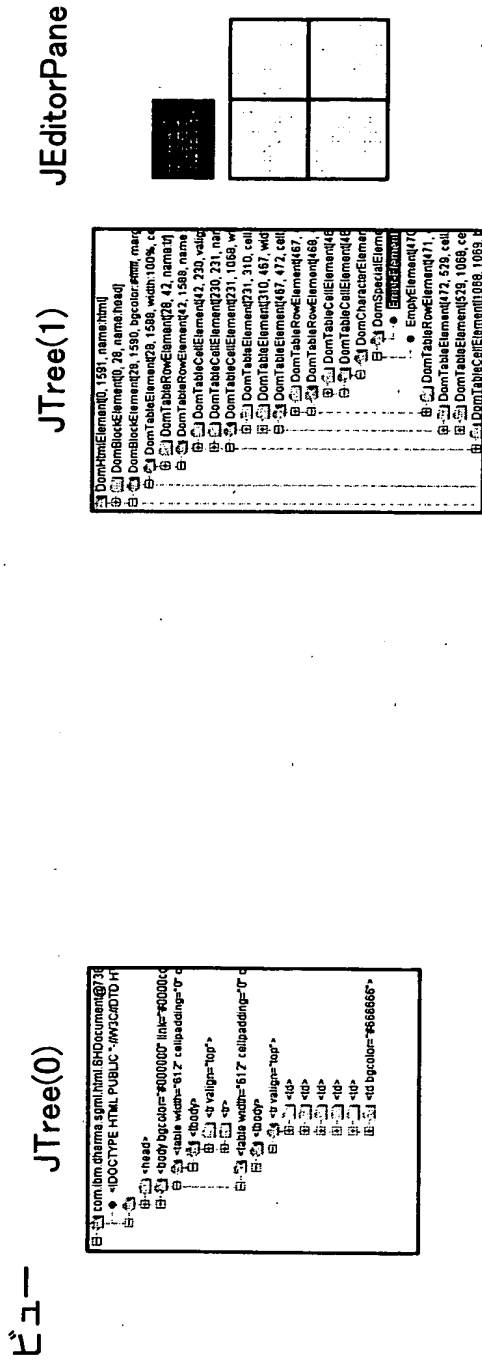
【図 17】



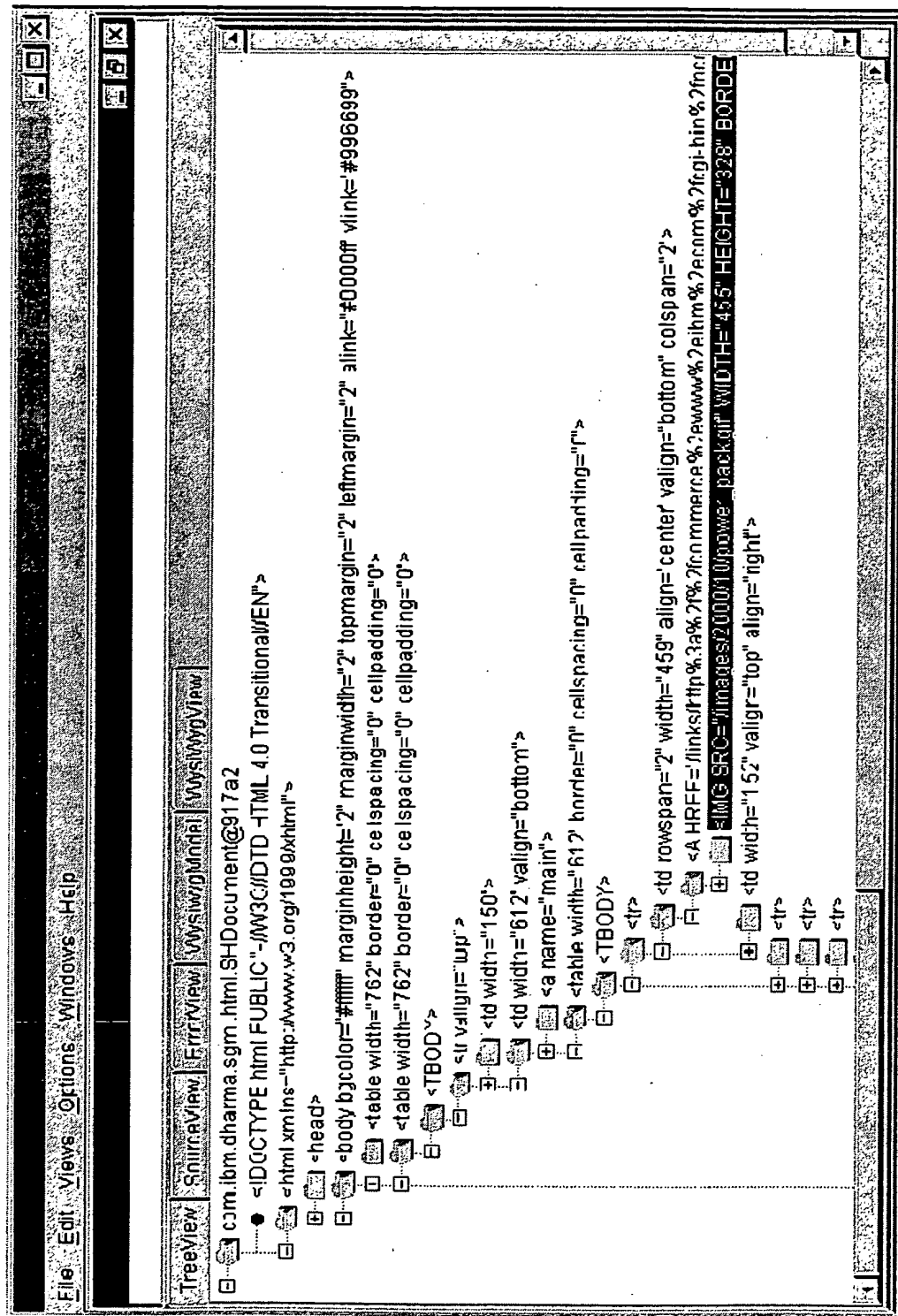
【図 18】



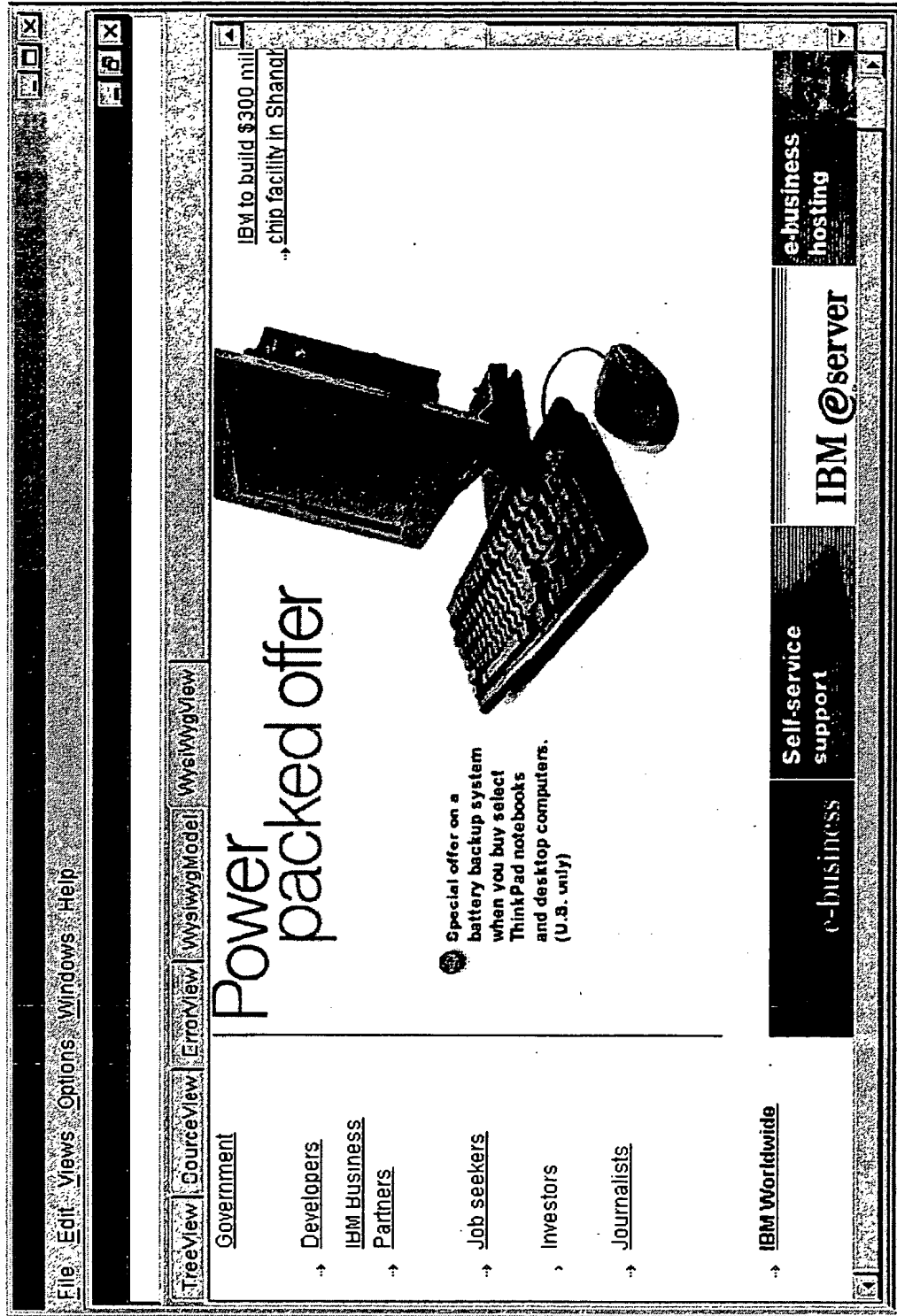
【図 19】



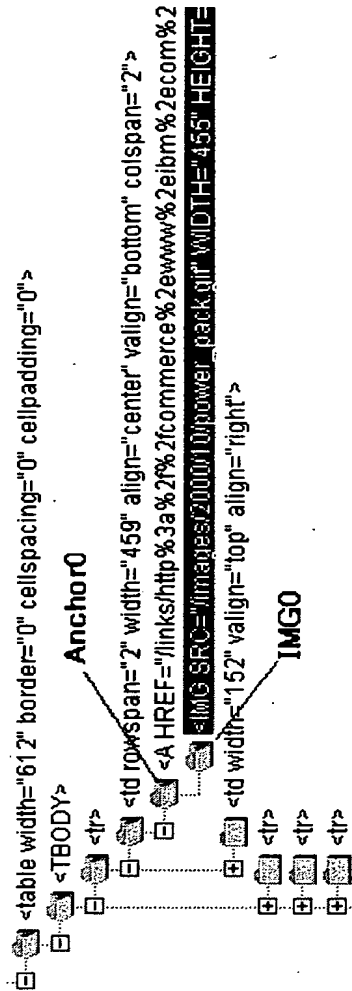
【図 20】



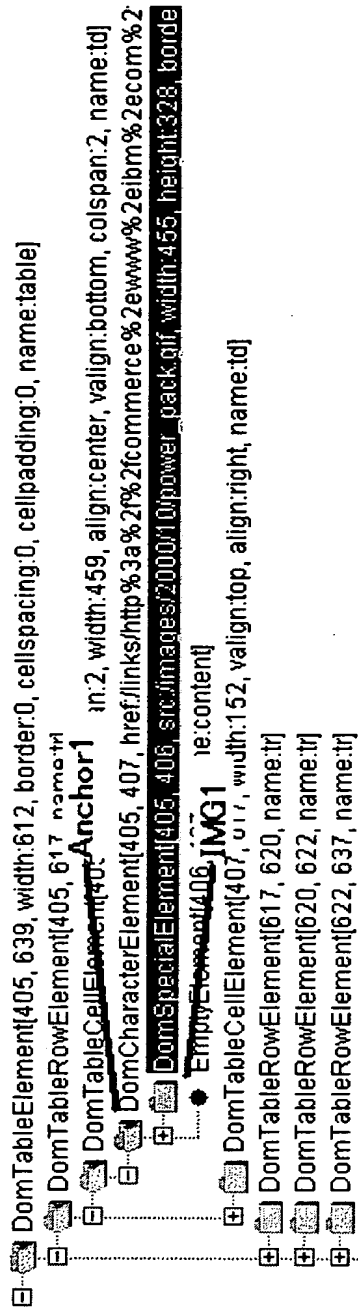
【図 2 2】



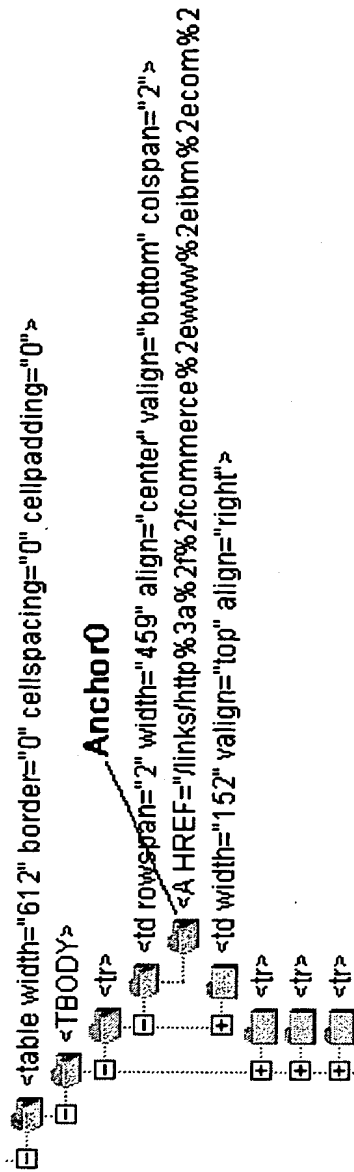
【图 2 3】



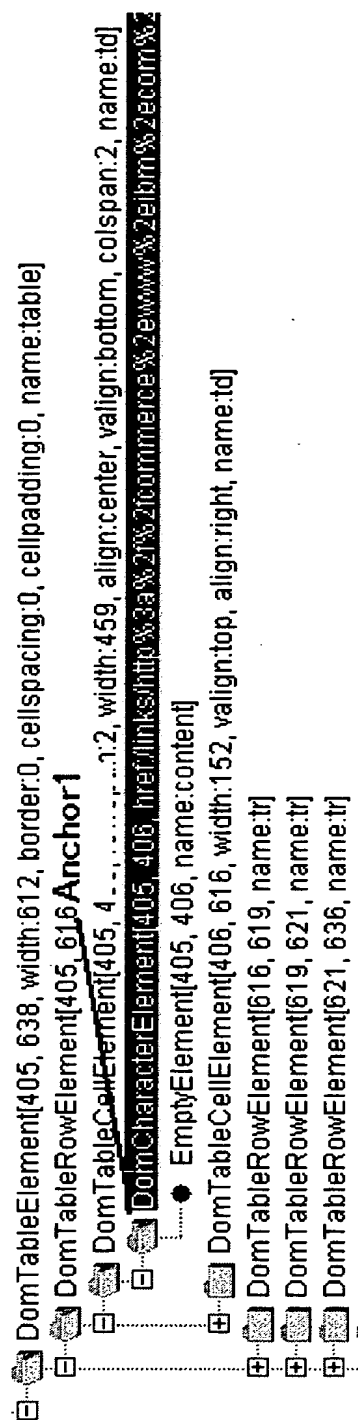
【图 2 4】



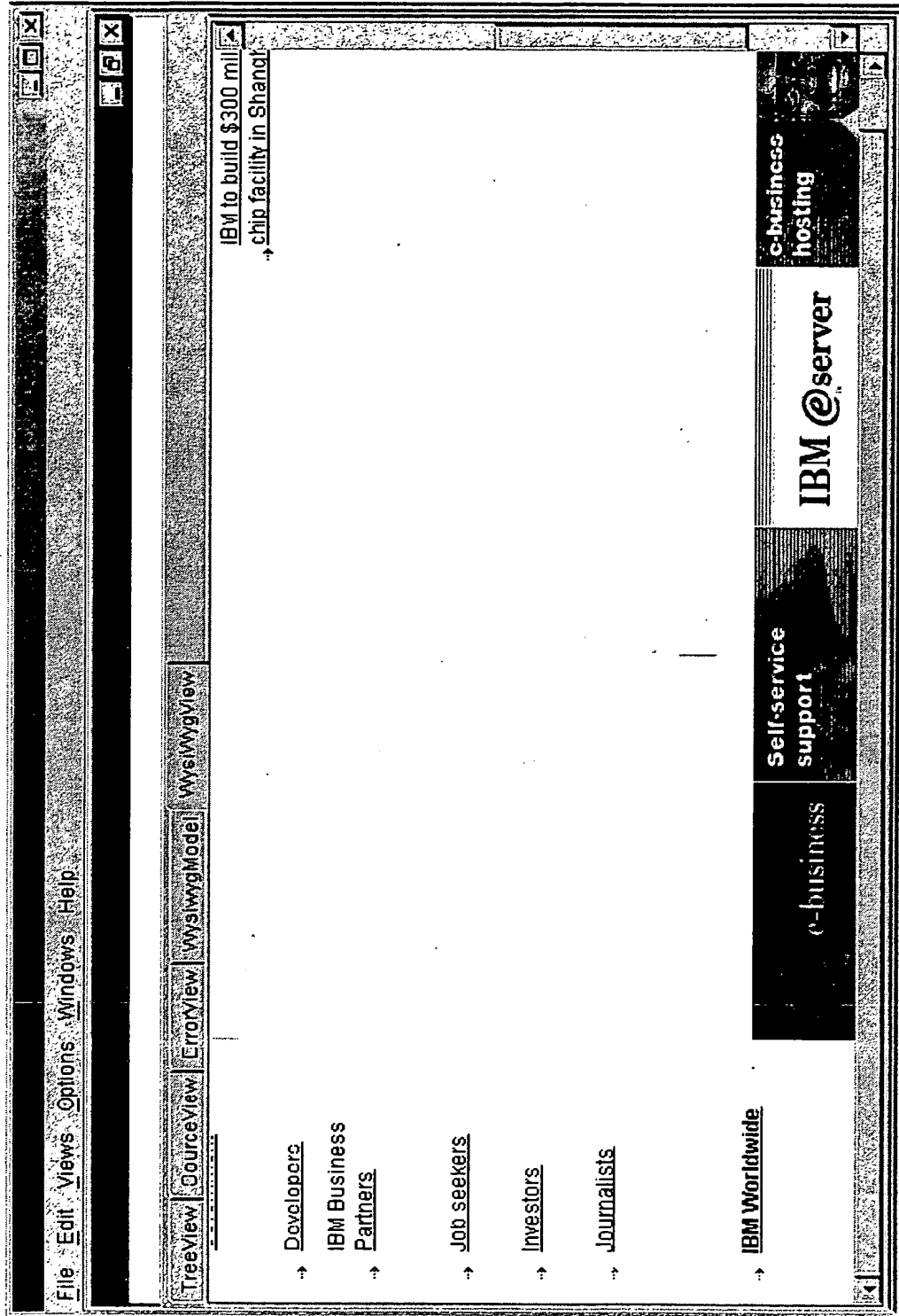
【図 2 5】



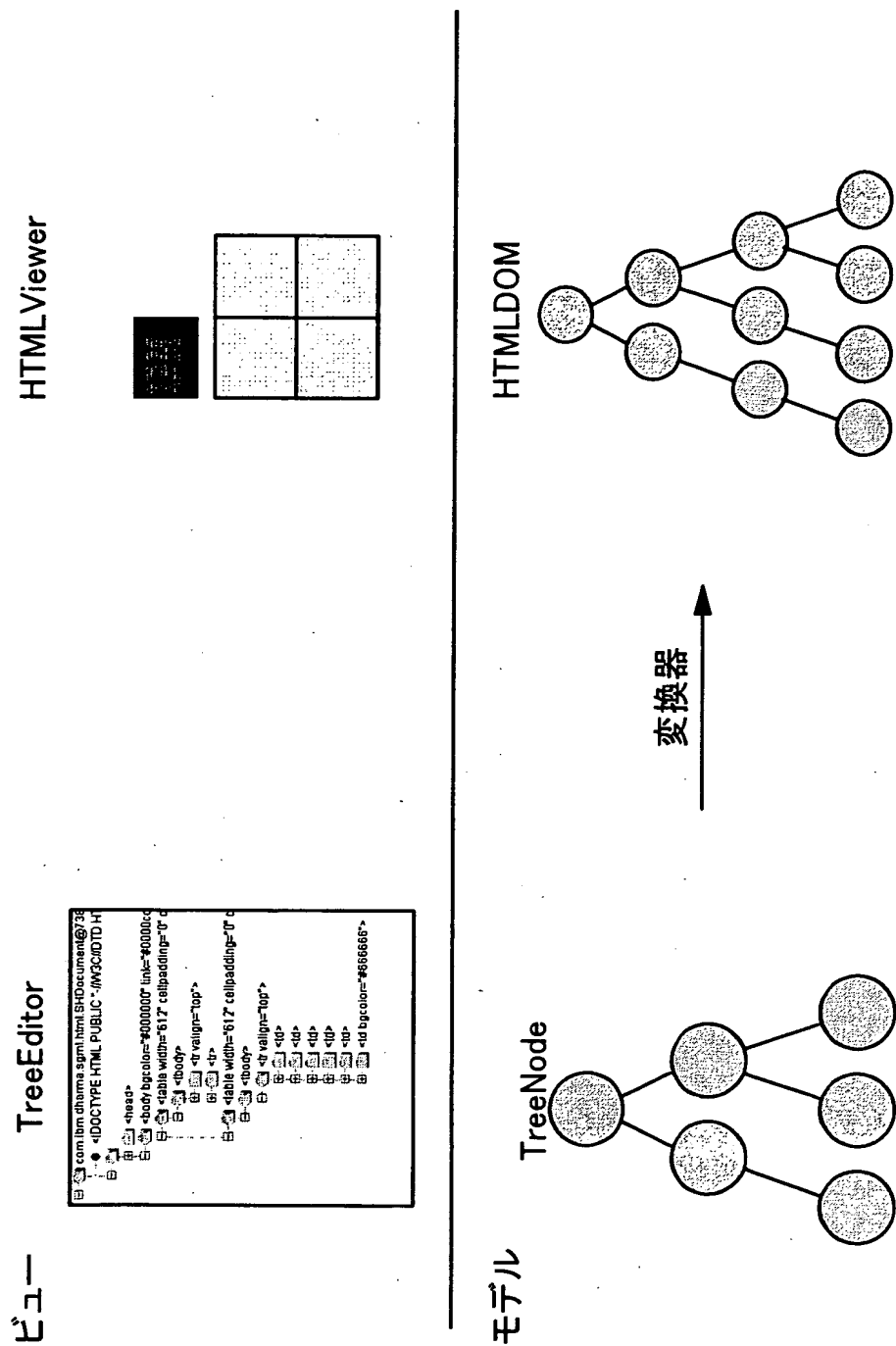
【图 2 6】



【図 27】



【图 28】



【図 2 9】

```

<?xml version="1.0"?>
<!ELEMENT discussion thread*>
<!ATTLIST discussion name CDATA #REQUIRED>
<!ELEMENT thread statement+>
<!ATTLIST thread id ID #IMPLIED>
<!ELEMENT statement (content, statement*)>
<!ATTLIST statement
  subject CDATA ""
  writtenby CDATA "anonymous"
  id ID #IMPLIED
>
<!ELEMENT content #PCDATA>
    
```

【図 3 0】

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:template match="discussion">
  <html>
    <head>
      <title>
        <xsl:valueof select="@name"/>
      </title>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
<xsl:template match="thread">
  <ul>
    <xsl:apply-templates/>
  </ul>
</xsl:template>
<xsl:template match="statement">
  <li><b><xsl:valueof select="@subject"/></b> - <i><xsl:valueof
select="@writtenby"/></i><br/>
    <xsl:apply-templates select="content"/>
  <ul>
    <xsl:apply-templates select="statement"/>
  </ul>
</li>
</xsl:template>
<xsl:template match="content">
  <xsl:apply-templates/>
</xsl:template>

```

【図 3 1】

```
<?xml encoding="US-ASCII"?>
<!DOCTYPE discussion SYSTEM "discussion.dtd">
<discussion name="Transformer Discussion">
  <thread id="1">
    <statement subject="Test" writtenby="kondo">
      <content>This is a test.</content>
    </statement>
  </thread>
</discussion>
```

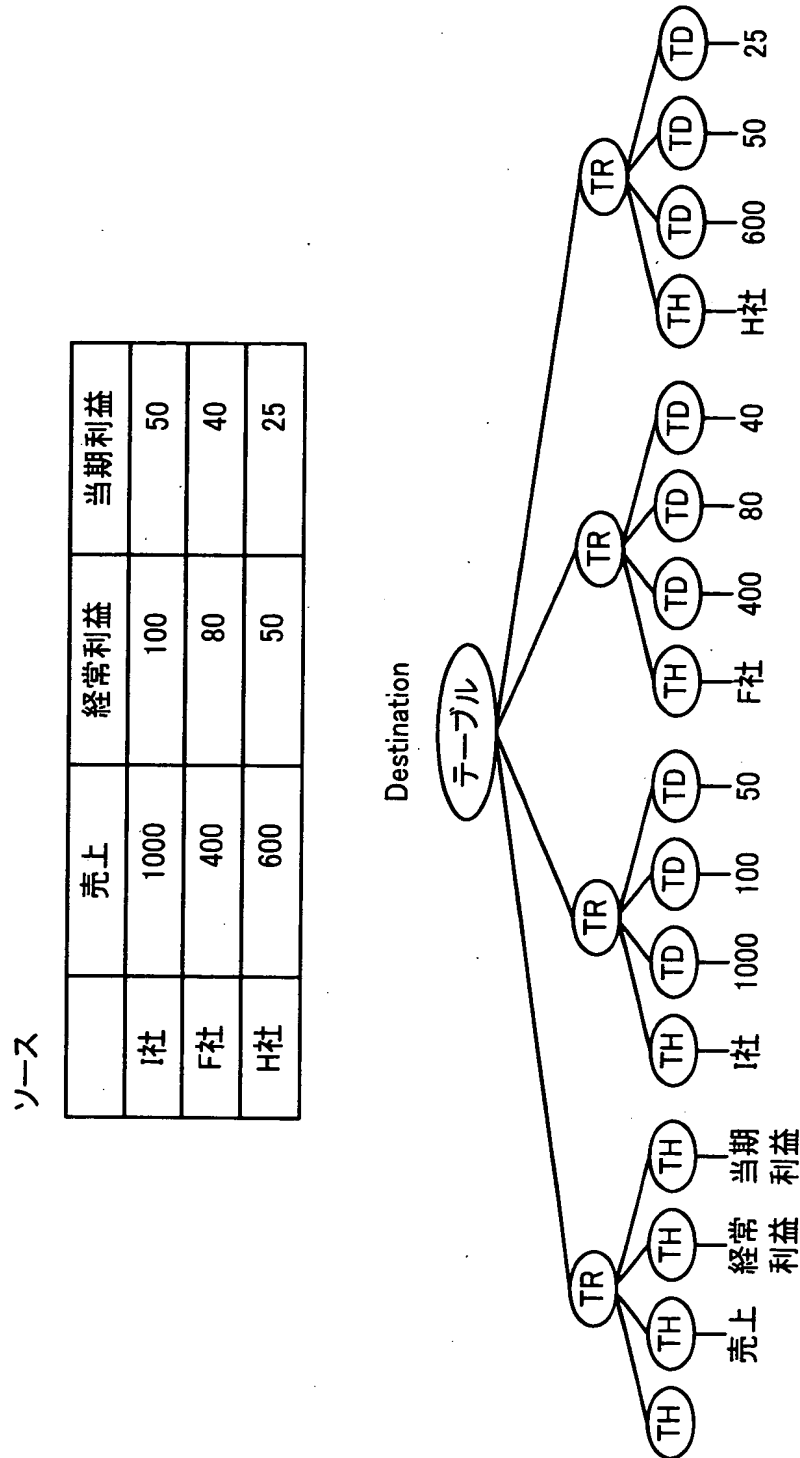
【図 3 2】

```
<html>
<head>
<title>Transformer Discussion</title>
</head>
<body>
<ul>
<li><b>Test</b> - <i>kondo</i><br>
This is a test.
<ul>
</ul>
</ul>
</body>
</html>
```

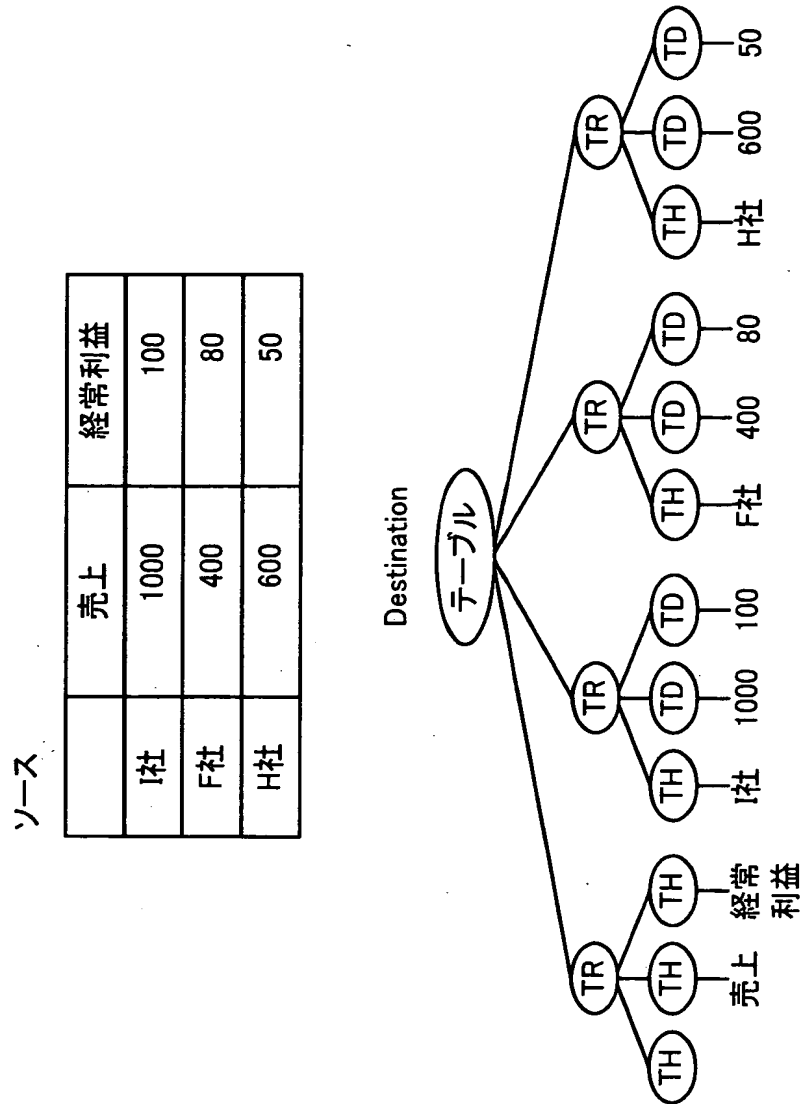
【図 3 3】

```
<html>
<head>
<title>Transformer Discussion</title>
</head>
<body>
<ul>
<li><b>Test</b> - <i>kondo</i><br>
This is a test.
<ul>
<li><b>OK</b> - <i>anonymous</i><br>
OK
</ul>
</ul>
</body>
</html>
```

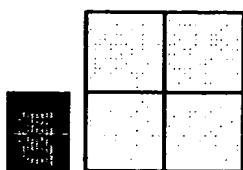
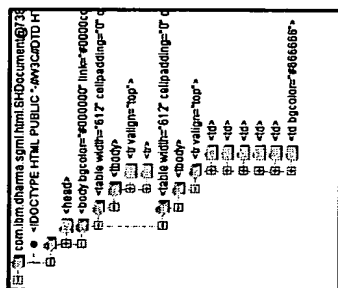
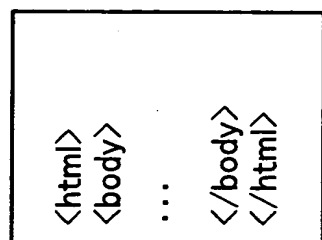
【図 3 4】



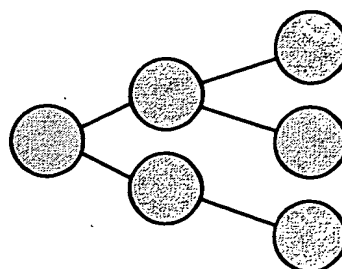
【図 3 5】



【图 3 6】

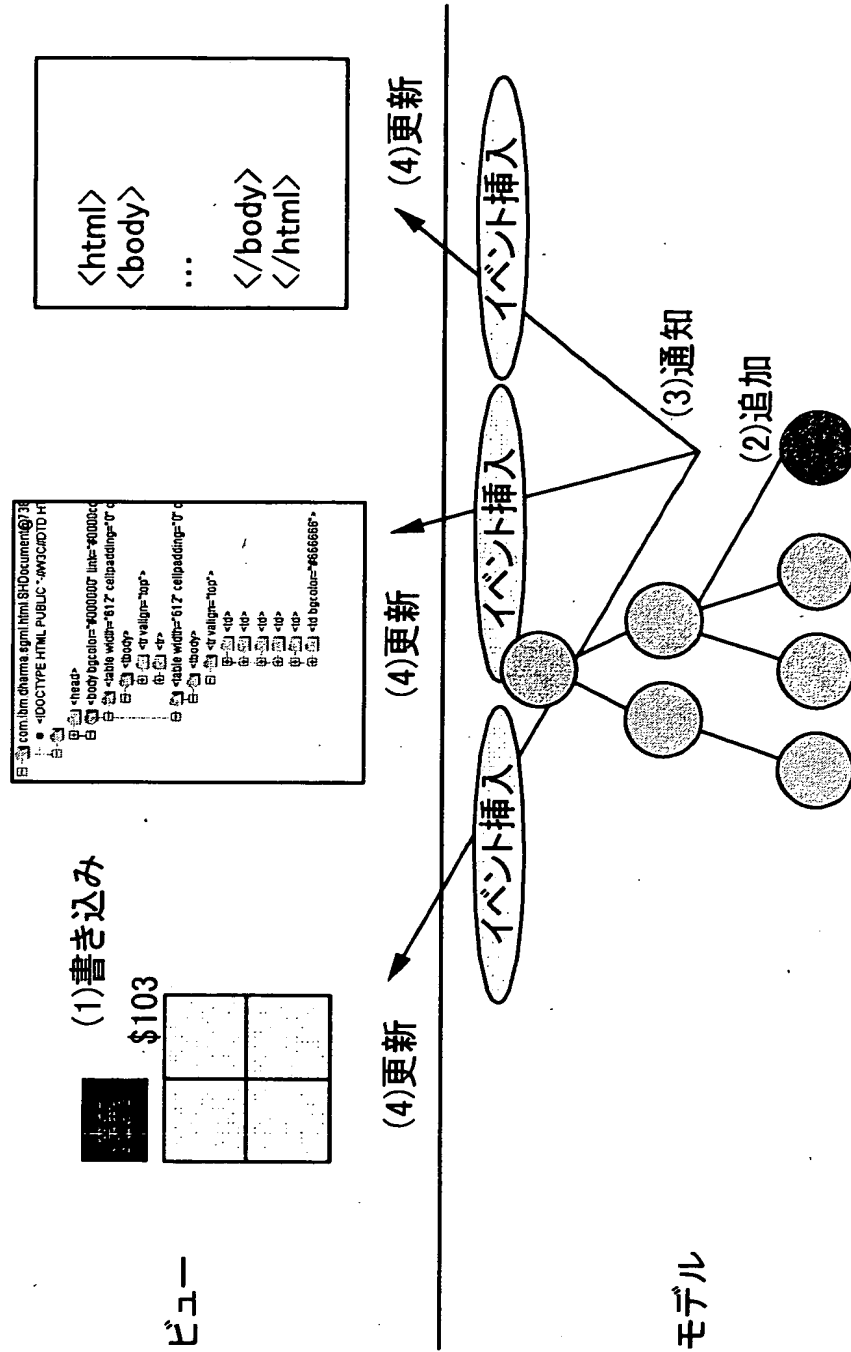


一七

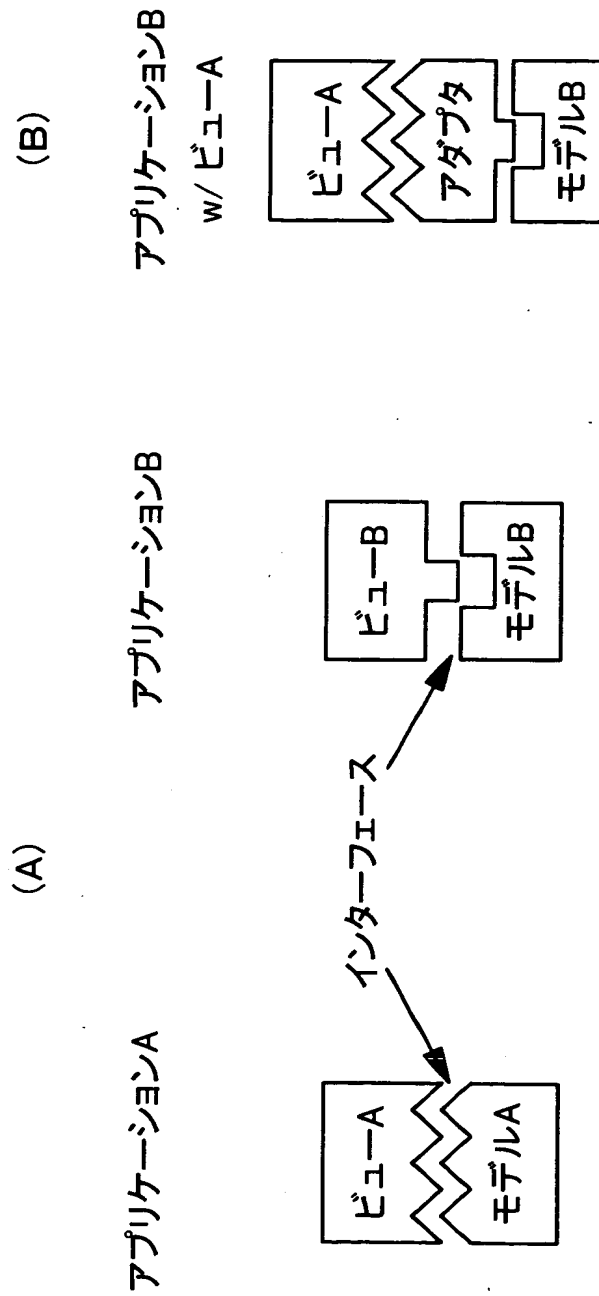


モデル

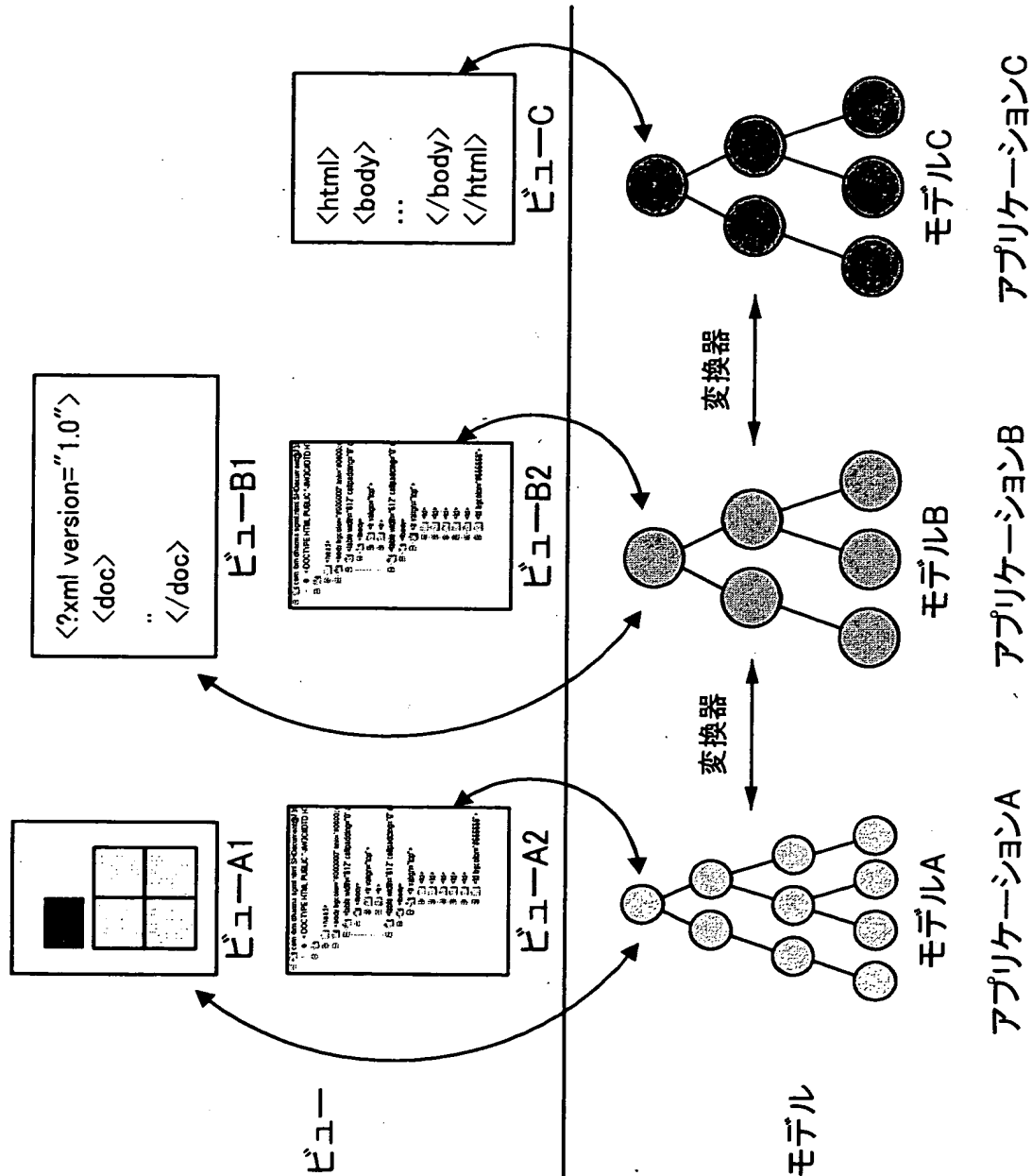
【図 37】



【図 3 8】



【図 39】



【書類名】 要約書

【要約】

【課題】 モデル変換を用いて所定のモデルから他のアプリケーションのビューを使用する場合に、変換元のモデルに対する変更を動的に変換先のモデル及びビューに反映させ、ビューを更新できるようにする。

【解決手段】 モデルとビューとを別個に有するアプリケーションの編集を行うアプリケーション編集装置において、このアプリケーションにおける第1のモデルを編集するアプリケーションA実行部10と、この第1のモデルを第2のモデルに変換するモデル変換器20と、この第2のモデルをそのビューにてディスプレイ装置に表示するアプリケーションB実行部30とを備える。そして、このアプリケーションB実行部30は、第2のモデルが更新された場合に、この第2のモデルの更新に基づくイベントを生成するイベント生成部32を備え、このイベント生成部32にて生成されたイベントに基づいてディスプレイ装置に表示されたビューを変更する。

【選択図】 図2

認定・付加情報

特許出願の番号	特願2001-246290
受付番号	50101197167
書類名	特許願
担当官	風戸 勝利 9083
作成日	平成13年 9月25日

<認定情報・付加情報>

【特許出願人】

【識別番号】	390009531
【住所又は居所】	アメリカ合衆国10504、ニューヨーク州 アーモンク (番地なし)
【氏名又は名称】	インターナショナル・ビジネス・マシーンス・コーポレーション

【代理人】

【識別番号】	100086243
【住所又は居所】	神奈川県大和市下鶴間1623番地14 日本アイ・ビー・エム株式会社 大和事業所内
【氏名又は名称】	坂口 博

【代理人】

【識別番号】	100091568
【住所又は居所】	神奈川県大和市下鶴間1623番地14 日本アイ・ビー・エム株式会社 大和事業所内
【氏名又は名称】	市位 嘉宏

【代理人】

【識別番号】	100106699
【住所又は居所】	神奈川県大和市下鶴間1623番14 日本アイ・ビー・エム株式会社大和事業所内
【氏名又は名称】	渡部 弘道

【復代理人】

【識別番号】	100104880
【住所又は居所】	東京都港区赤坂5-4-11 山口建設第2ビル 6F セリオ国際特許事務所
【氏名又は名称】	古部 次郎

【選任した復代理人】

【識別番号】	100100077
--------	-----------

次頁有

認定・付加情報（続き）

【住所又は居所】 東京都港区赤坂5-4-11 山口建設第2ビル
6F セリオ国際特許事務所
【氏名又は名称】 大場 充

出 願 人 履 歴 情 報

識別番号 [390009531]

1. 変更年月日 2000年 5月16日

[変更理由] 名称変更

住 所 アメリカ合衆国10504、ニューヨーク州 アーモンク (番地なし)

氏 名 インターナショナル・ビジネス・マシーンズ・コーポレーション